COMPUTATIONAL APPROACHES TO INTUITIVELY ANALYZE AND VISUALIZE SINGLE-CELL DATA

A Dissertation
Presented to
The Academic Faculty

By

Xingyu Yang

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Biology

Georgia Institute of Technology

May 2019

COMPUTATIONAL APPROACHES TO INTUITIVELY ANALYZE AND VISUALIZE SINGLE-CELL DATA

Approved by:

Dr. Peng Qiu, Advisor
Department of Biomedical Engineering
Georgia Tech and Emory University

Dr. Soojin Yi
School of Biological Sciences
Georgia Institute of Technology

Dr. Gregory Gibson
School of Biological Sciences
Georgia Institute of Technology

Dr. David Archer
Department of Pediatrics
Emory University

Dr. Ignacio Sanz
Department of Medicine
Emory University

Date Approved: December 18, 2018

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

In this thesis, I will focus on developing computational methods that deliver intuitive and interpretable visualization of single-cell data. The first project describes a software named Cluster-to-Gate (C2G) that can visualize existing clustering results of flow/mass cytometry data in the format of 2D gating hierarchy. Though C2G presents a way to visualize and interpret clustering results, the visualization is still data-driven and does not involve human-knowledge. To overcome the limitation of C2G, the following project describes a framework that can learn gating approach from existing publications to build a knowledge-graph. This knowledge-graph can automatically suggest the order of marker usage and gating hierarchy for the new data set, which can be used to gate the cell population. The obtained cell populations are immediately matched to some cell types in the knowledge graph, which makes them more interpretable. The last project describes a novel algorithm (GLaMST) to reconstruct lineage tree of B-cell receptor genes from high throughput sequencing data. This algorithm outperforms state-of-art in both accuracy and speed.

CHAPTER 1
INTRODUCTION

To understand the secret of life and improve human health, great efforts and resources have been invested in biological research. Modern biology focuses on the understanding of cells and cell interactions in molecular level [1].To quantitatively understand the cell behaviors in molecular level, many technologies are conducted by pooling multiple cells together and report average behavior of a huge number of cells. Such average measurement is shown to be limited in exploring the cellular heterogeneity within isogenic cell population [2, 3, 4]. Such heterogeneity is known to be the outcome of stochasticity in gene expression [5], which reflects cell type composition and determines cell fate [6, 7]. One study in single-cell gene expression gives an example that the average expression of 50 cells does not reflect any of the individual cells[3]. Another study on the correlation between single-cell gene expression and known cell lineages in *Caenorhabditis elegans* shows that even within the homogeneous tissue, individual cells show clear heterogeneity[4]. In response to the observations above, researchers have developed plenty of single-cell resolution technologies to better understand cell diversity and heterogeneity.

Cytometry is a single-cell resolution technique that can measure the expressions of multiple protein markers and provide data to reveal cellular heterogeneity and sub-populations [8, 9]. The typical number of protein markers is up to 12 for flow cytometry [10], and up to 50+ for CyTOF [9]. Such single-cell data enables quantification of cellular heterogeneity [11, 12], discovery of novel subpopulation [13, 14] , identification of rare cell types [15], and correlation between single-cell characteristics and clinical features [16, 17].

Due to the recent advancement in flow and mass cytometry, researchers can quickly generate massive and high-dimensional datasets. Such huge data can no longer be analyzed by traditionally manual gating [18], where gates are manually drawn on user-defined

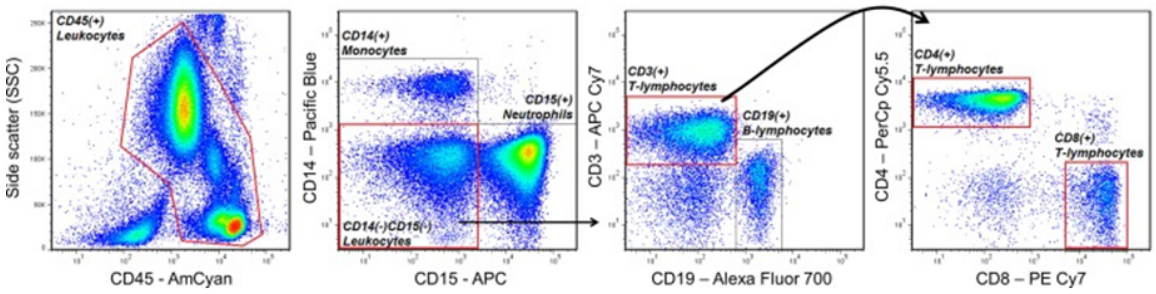

Figure 1.1: **Example of Manual Gating** In this example, each panel is a scatter plot of two markers. The dots in the scatter plot represent cells and their color means the local density. The red and black polygons in the image represent gates drawn by human experts. Cells in the red gates are of interest and will be used in drawing the scatter plots in the next panel.

sequences of nested two-dimensional plots as the example shown in Figure 1.1. To overcome the limitation of manual gating, many automated algorithms are developed to analyze high dimensional data [19, 20, 21]. Most of the current automated algorithms in flow/mass cytometry data are data-driven. Data-driven analysis usually suffers from the following issues. First, the data analysis results are not directly interpretable. They usually can only tell the number of cell populations exists in the dataset but cannot match these populations to meaningful cell types. Second, traditionally, biologists understand cell populations identified in flow/mass cytometry data in a hierarchical view. For example, T cells and B cells are sub-cell-type of lymphocytes. Data-driven methods cannot obtain such hierarchical structure without additional human input. Third, data-driven methods usually suffer from unbalanced data size, especially rare cell populations.

Motivated by the first difficulty, several visualization methods (e.g., viSNE[22], SPADE[11]) have been developed to nonlinearly project high-dimensional data into two dimensions, creating two-dimensional visualizations that can help biologists to interpret the data. Although these nonlinear visualizations provide rich information about the high-dimensional relationship in the data, they are not intuitive to biologists who are accustomed to the two-dimensional nested gating representations. To derive intuitive interpretations of high-dimensional cytometry data analysis, several methods have been developed for various purposes. For example, flowType [23] and RchyOptimyx [24] were designed to enumerate

all cell types defined by clustering algorithms, and identify efficient biomarker combinations that correlate with clinical features. AutoGate[25] was designed to automatically draw gates in an unsupervised fashion, producing gating visualizations without requiring users to manually define boundaries of the gates. Implementations in OpenCyto[26] include automated pipelines for generating gating visualizations given a user-defined gating hierarchy, which can be viewed as a supervised approach.

In **Chapter Two** of my thesis, I intended to solve the interpretation problem by generating a gating hierarchy and conventional gating plots for one or more target cell populations, defined by high-dimensional clustering algorithms. Given a cytometry dataset and some clustering analysis that defines one or multiple target populations of interest, I developed Cluster-to-Gate (C2G) to generate a gating hierarchy that captures the target populations, and present the hierarchy in nested two-dimensional gating sequences that resemble the conventional manual gating analysis. The gating hierarchy is constructed using an entropy-based approach to select marker pairs, and polygon-shaped gates are drawn based on overlaps among the target populations of interest and the unlabeled cells. With this approach, automated clustering algorithms can present the clustering results in gating visualizations that biologists are familiar with, making the clustering results more intuitive and easier to interpret. C2G can also benefit algorithm developers by providing gating visualizations to assist the debugging and performance evaluation of their algorithms. In addition to visualization and interpretation, C2G can be used in panel refinement by providing a relative importance measurement for different protein markers.

Despite such applications, the visualization provided by C2G is still data-driven and not guaranteed to provide a biologically meaningful hierarchical structure. To address this issue, I propose to analyze the cytometry data in a knowledge-assisted way. However, for different marker settings, the prior knowledge required to analyze the cytometry is very different. To automatically select the right knowledge for different marker settings, I need to build a knowledge graph that standardizes knowledge from different sources. In **Chapter**

Three, I will describe a framework that can learn knowledge from existing manually gated flow/mass cytometry and automatically generate gating strategy for new data with different experiment settings. Viewing each manual gating strategy as a hierarchical tree, I merge different hierarchical trees from different publications into one graph by matching their cell types and marker usage. Given the marker settings of a different dataset, the graph can automatically suggest gating strategy by picking up sub-trees that covering most of the given markers. Gating strategies generated in this way naturally have a biologically meaningful hierarchical structure.

In addition to flow and mass cytometry, the recent development of next-generation sequencing (NGS) enables researchers to conduct single-cell resolution study of nucleic acids[27, 28]. Such single-cell sequencing brings new insight into many applications including new cell type identification, inferring regulatory networks, and cell hierarchy reconstruction[29]. **Chapter Four** of my thesis mainly deals with the computational problem in cell hierarchy reconstruction of lineages of B cell receptor gene. To specifically recognize and respond to different pathogens, adaptive immune system relies on a diverse repertoire of B cell receptors (BCR). Such a diverse repertoire comes from recombination, somatic hypermutation of immunoglobulin (Ig) gene segments, and selection by their ability to bind pathogens. This process will generate numerous different BCRs. To explore the dynamic process of BCR affinity maturation, researchers have applied high throughput sequencing [30, 31, 32] to examine BCR repertoires and to construct lineage trees of BCR sequences [33, 34].

In a BCR lineage tree, each tree node corresponds to one unique sequence, and each directed edge indicates the relationship between one sequence and its immediate ancestor, which are separated by one-base mutation, insertion or deletion. Given high throughput BCR sequencing data of a repertoire, the observed sequences correspond to some of the internal nodes and the leaf nodes of the BCR lineage tree, while many intermediate nodes are not observed due to the diversification and selection process the repertoire went through,

as well as subsampling inherent to the assay. With these observed sequences, we can easily identify the root sequence of this tree by sequence alignment against known germline BCR segments in the genome [35]. To reconstruct the full lineage tree, we need to fill in the unobserved internal nodes and connect them to the observed nodes by direct edges. This process is similar to building the phylogenetic tree among species, except that only leaf nodes are observed in the phylogenetic problem, whereas some internal nodes and the root nodes are also observed in this BCR lineage tree reconstruction problem. Popular methods in the phylogenetic analysis include maximum parsimony, maximum likelihood, and Bayesian methods. The maximum likelihood and Bayesian methods are usually computational demanding and require a decent amount of prior knowledge [36, 37], for example, the replacement rates and preference of mutation target under selection pressure [38]. Such prior knowledge is relatively limited in BCR lineage trees. Therefore, we decided to pursue the maximum parsimony idea to reconstruct the BCR lineage tree, which intends to reconstruct a tree as small as possible, which connects all the observed sequences with the minimum number of mutation, insertion and deletion events.

Reconstruction of a phylogenetic tree using the maximum parsimony method is known to be a non-deterministic polynomial-time complete (NP-complete) problem [39, 40], which means no guarantee of the best solution in polynomial time. In terms of computational complexity, reconstruction of BCR lineage tree is very similar to a phylogenetic tree. Although the root sequence and some of the internal nodes are known, it is still an NP-Complete problem [34]. To reconstruct BCR lineage trees from high throughput sequencing data, an algorithm named IgTree was previously developed, which is a heuristic procedure consisting of multiple components [34]. It first constructs a preliminary tree that only contains observed sequences based on multiple sequence alignment [41], then uses a complex scoring metric to gradually add internal nodes to complete the full tree, and finally scans the resulting tree to identify subtrees that can be reduced by reversion events. IgTree enabled efficient analysis of large BCR sequence datasets, and brought insights into various area

of somatic-hypermutation-related biological processes including neutralization of HIV antibodies [42] and progression of follicular lymphoma [43]. Another algorithm for reconstructing BCR lineage trees is included in the TIgGER software package [44], which is a classical maximum parsimony method called ”dnapars” in the PHYLIP library [45, 46]. dnapars almost always achieves smaller lineage trees compared to IgTree but is considerably slower when analyzing a large number of observed sequences. My solution to such computational problem is a new algorithm named GLaMST. It reconstructs BCR lineage trees using the maximum parsimony criterion. GLaMST uses simple heuristics to Grow the Lineage trees along the Minimum Spanning Tree. In terms of the maximum parsimony criterion, GLaMST generates lineage trees with a small size similar to dnapars and outperforms dnapars for simulated datasets with insertions and deletions. In terms of the computational efficiency, GLaMST runs faster than IgTree.

CHAPTER 2

CLUSTER-TO-GATE

2.1 Introduction

Cytometry is a popular technique with wide applications in biology and medicine. It can measure the expressions of multiple protein markers for a large number of cells, and provide data to reveal cellular heterogeneity and subpopulations [8, 9]. Traditionally, flow cytometry data is analyzed by manual gating, where gates are manually drawn on user-defined sequences of nested two-dimensional plots. Manual gating is a subjective and labor-intensive process, especially when the number of protein markers grows large [18].

Recently, new analysis algorithms [19, 20, 21] are developed to cluster cytometry data in high-dimensional space. However, it is a nontrivial challenge to present the high-dimensional clustering results in a way that is easy to understand and interpret. Motivated by this challenge, several visualization methods (e.g., viSNE[22], SPADE[11]) have been developed to nonlinearly project high-dimensional data into two dimensions, creating two-dimensional visualizations that can help biologists to interpret the data. Although these nonlinear visualizations provide rich information about the high-dimensional relationship in the data, they are not intuitive to biologists who are accustomed to the two-dimensional nested gating representations. To derive intuitive interpretations of high-dimensional cytometry data analysis, several methods have been developed for various purposes. For example, flowType[23] and RchyOptimyx[24] were designed to enumerate all cell types defined by clustering algorithms, and identify efficient biomarker combinations that correlate with clinical features. AutoGate[25] was designed to automatically draw gates in an unsupervised fashion, producing gating visualizations without requiring users to manually define boundaries of the gates. Implementations in OpenCyto[26] include automated

pipelines for generating gating visualizations given a user-defined gating hierarchy, which can be viewed as a supervised approach.

In this research, I aimed to address a different supervised visualization challenge: how to automatically generate a gating hierarchy and conventional gating plots for one or more target cell populations, defined by high-dimensional clustering algorithms. This is a challenging problem because clustering algorithms often generate clusters that cannot be sufficiently characterized by combinations of positive or negative expression of protein markers, due to the high-dimensional nature of the data. Meanwhile, answering this problem can be highly useful, because it can generate conventional gating visualizations for automated clustering algorithms, assisting the biological interpretations of cell populations identified by the clustering algorithms.

To bridge the gap between the high-dimensional data analysis and conventional gating visualizations, we present a new method called Cluster-to-Gate (C2G). Given a cytometry dataset and some clustering analysis that defines one or multiple target populations of interest, C2G is able to generate a gating hierarchy that captures the target populations, and present the hierarchy in nested two-dimensional gating sequences that resemble the conventional manual gating analysis. The gating hierarchy is constructed using an entropy-based approach to select marker pairs, and polygon-shaped gates are drawn based on overlaps among the target populations of interest and the unlabeled cells. With this approach, automated clustering algorithms can present the clustering results in gating visualizations that biologists are familiar with, making the clustering results more intuitive and easier to interpret. C2G can also benefit algorithm developers by providing gating visualizations to assist the debugging and performance evaluation of their algorithms. We applied C2G to examine both simulated data and previously published mass cytometry datasets [47, 48] to demonstrate its performance and robustness in visualizing target populations defined by manual gating, automated clustering algorithms (k-means, flowMeans[49], flowSOM[50] and PhenoGraph[51]), and visualization-assisted methods (SPADE[11] and tSNE[22]).

2.2 Method

2.2.1 General Workflow

Inputs of C2G are single-cell data generated by flow cytometry or mass cytometry, and cell labels indicating which cells belong to the pre-defined target populations of interest. Flow and mass cytometry data are typically stored in the FCS format [52] which contains intensity measurements of multiple protein markers across a large number of cells. Since the input cell labels only cover target populations of interest, which do not necessarily cover all the cells in the data, we first pre-cluster the unlabeled cells based on their overlap with target populations. Then, we use an entropy-based approach to iteratively generate a gating hierarchy. The iterative algorithm starts from the root node of the gating hierarchy, which contains all cells in the data. We automatically derive gates based on all possible marker pairs and develop an entropy measure to evaluate their ability in separating the target populations and the pre-clustered unlabeled cells. Gates derived based on the best marker pair define cell subsets, which form children nodes of the root node and are subjected to further gating iterations. This process repeats until no marker pair can further separate any target populations in any node (Figure 2.1). The output of the algorithm is a tree representation of the gating hierarchy, and two-dimensional plots showing the selected gates. To illustrate this method, we generated a simulated dataset in three dimensions with five cell populations, and the two colored populations are treated as the target populations of interest (Figure 2.2a). Given such inputs, C2G automatically generated a gating hierarchy to gate for the two target populations.

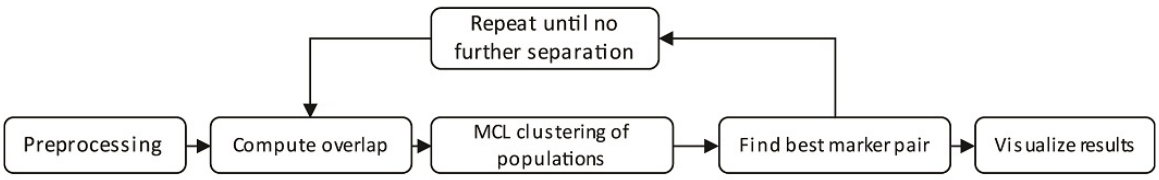

Figure 2.1: **General workflow of C2G**

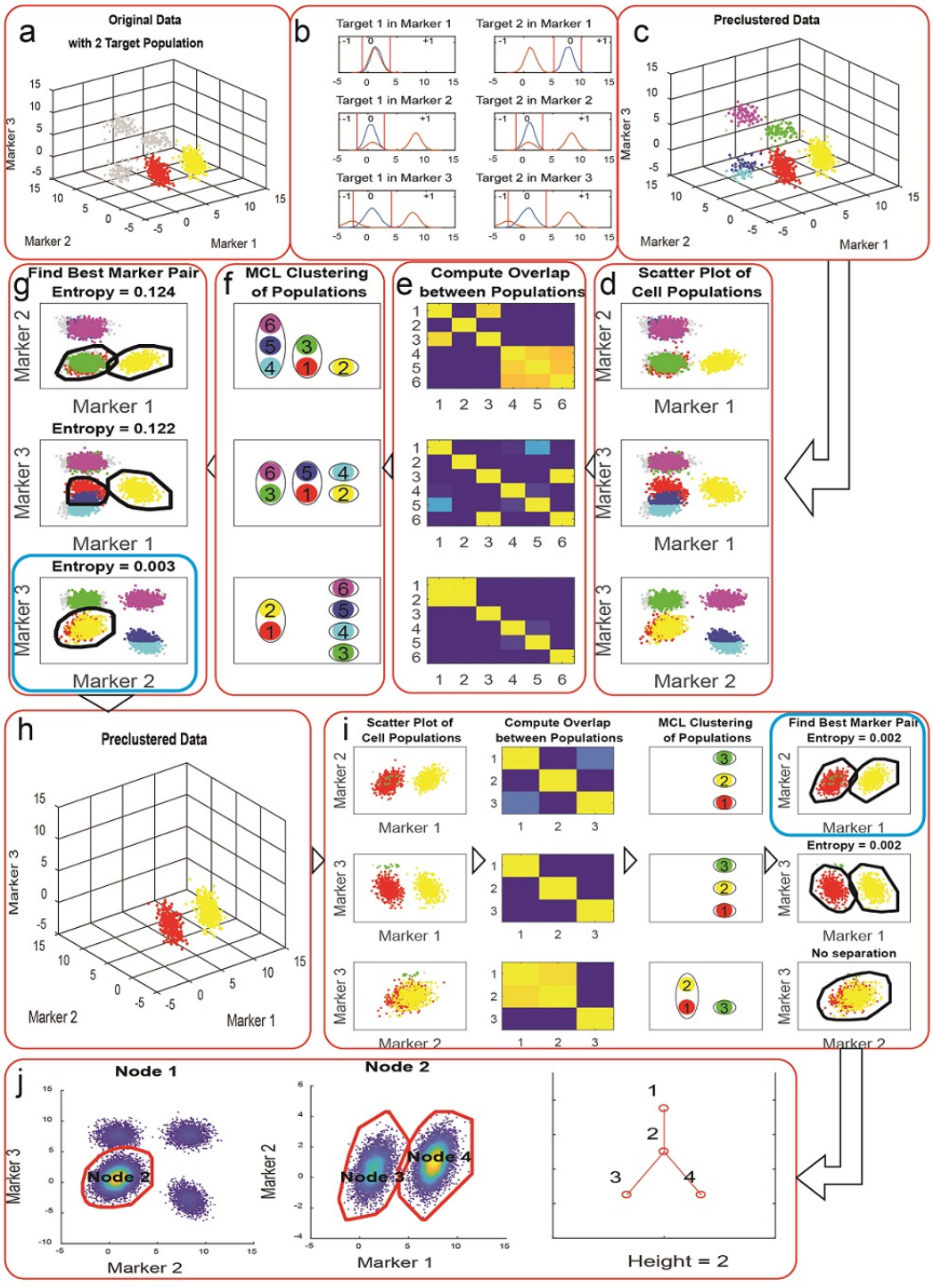

Figure 2.2: **C2G applied to a simulated illustrative example**

Figure 2.2: (a) Simulated data. Target populations are colored as red and yellow. The two target populations cannot be directly separated from the unlabeled cells and each other in the 2D biaxial plot of any marker pair. (b) Pre-cluster the unlabeled cells. Each subplot shows two distributions to visualize how the unlabeled cells overlap with a target population according to a marker. The blue curve shows the distribution of the marker expression in the target population, and the red curve shows the marker expression distribution in the unlabeled cells. Two red vertical lines show the threshold we choose to characterize unlabeled cells according to the target population and the marker. Unlabeled cells located between the two vertical lines are assigned a value of 0, whereas unlabeled cells located to the right or left side of the two vertical lines are assigned a value of +1 or -1. (c) Pre-clustering result. Each pre-clustered unlabeled population is shown with a separate color. (d) Draw 2D biaxial scatter plot of each possible marker pair. (e) Compute overlap between all target and pre-clustered unlabeled populations. (f) Apply MCL algorithm to cluster the populations. Populations clustered together are in the same eclipse. (g) Automatically draw gates on each marker pair. The black polygons show the gates based on the population clusters derived by MCL. The entropy on top of each plot quantifies the quality of the gates, in terms of their ability to separate target populations from unlabeled cells, and from each other. The lower entropy the better gating quality. The blue rectangle shows the best marker pair. (h) Cells obtained after the first gate. (i) Derive further gates by repeating steps (c)-¿(g) to cells in the first gate. (j) The final output of C2G. First two figures show the gating sequence, and the third figure is a tree structure that represents the gating hierarchy.

2.2.2 Pre-Cluster the Unlabeled Cells

When we are only interested in a subset of the populations in the data, the input labels only cover the cells belonging to the populations of interest, and the remaining cells are considered as unlabeled. It is essential to pre-cluster the unlabeled cells so that we can develop an effective measure to evaluate the overlap and separation among the target populations and the unlabeled cells. We pre-cluster the unlabeled cells based on their marker expression in relation to the target populations. Figure 2.2b shows an example with two target populations and three markers. For target population 1 and marker 1, we define two thresholds based on the 1st and 99th percentile of the marginal distribution of the markers expression in the target population. These two thresholds can be used to define one feature to characterize an unlabeled cell: is marker 1 expression of the unlabeled cell lower (-1), similar (0), or higher (-1) than target population 1. By examining all target populations and all protein markers in this example, an unlabeled cell is characterized by 6 features with values -1/0/1,

showing whether its expressions of the three markers are low/similar/high with respect to the two target populations. Pre-clustering is performed by grouping unlabeled cells with the same values in all features into the same cluster. Since outliers in the data will form clusters with the very small number of cells, we discard clusters whose sizes are smaller than a certain threshold (e.g., 5 cells). In the example shown in Figure 2, the unlabeled cells (gray in Figure 2.2a) are pre-clustered into four clusters shown in Figure 2.2c. Although the pre-clustering method may not be perfect (e.g., the bottom-left unlabeled population is divided into two clusters), it is sufficient for the purpose of measuring the separation between target populations and unlabeled cells.

2.2.3 Draw Best Gates for Each Marker Pair

Given a set of cells (represented by one node in the tree of gating hierarchy) and a marker pair (which define a biaxial plot on which gates can be drawn), we would like to automatically generate gates that are able to separate as many populations as possible, both the target populations and the pre-clustered populations from the unlabeled cells. In the 2D space defined by the marker pair, we first compute the overlap between all pairwise combinations of the populations. We partition the 2D space into 40-by-40 grids and count the number of cells from each population falling into each grid. The overlap between populations i and j is defined as $\sum_{all\ grids\ k} \sqrt{q_{ik} q_{jk}}$ where q_{ik} and q_{jk} are the percentages of cells in populations i and j that fall into grid k. This strategy for computing overlaps is scalable to handle large cytometry datasets with millions of cells. Figure 2.2d shows the biaxial plots of the 6 target and pre-clustered populations for all three possible marker pairs in the example dataset, and Figure 2e visualizes the computed overlaps between each pair of populations.

After obtaining the overlap measurement between each pair of populations, we cluster the populations using the Markov Cluster (MCL) Algorithm [21], which is a graph partition algorithm based on simulation of stochastic flow in graphs. The input of the MCL algo-

rithm is an adjacent matrix that contains the pairwise overlap measurements. The MCL algorithm clusters the populations into groups with small overlap across the groups and automatically determines the appropriate number of groups. The clustering results of MCL is shown in Figure 2f. For each group that contains one or more target populations, a gate is drawn based on the union of the convex hulls of all populations in this group (Figure 2.2g). Groups containing only pre-clustered unlabeled populations are ignored. Therefore, for each marker pair, we derive one or a few gates, where each gate covers one or multiple target populations.

2.2.4 Identify Best Marker Pair

After deriving gates for each possible marker pair, we evaluate and determine the best marker pair, where the corresponding gates (1) best separate the target populations and (2) exclude as many pre-clustered unlabeled populations as possible. These two objectives may not always align with each other. For example, in the top panel of Figure 2.2g, two gates are drawn to separate the two target populations, and three of the four unlabeled populations are excluded. In the bottom panel of Figure 2.2g, only one gate is drawn, which excluded all four unlabeled populations, but the gate encompasses the two target populations and does not separate them.

To balance these two objectives in determining the best marker pair, we developed an entropy-based metric. For each marker pair and the gate(s) drawn in the 2D space defined by this marker pair, we count the effective number of cells in each gate from each population. If one cell falls into the intersection of k gates, it is counted as 1/k toward each of the k gates. The collection of all unlabeled cells is considered as one population, and the complement of the union of all gates is considered as one gate. Therefore, we obtain an (m+1)*(n+1) matrix of effective counts, where m is the number of target populations and the +1 is the unlabeled cells, n is the number of gates and the +1 represent the complement of the union of all gates, which is itself a gate. We normalize each row of the effective

counts matrix by the row sum and obtain a matrix of conditional probabilities p_{ij}, indicating the probabilities of cells from population i falling into gate j. To evaluate the marker pair, or more precisely the gates drawn based on the marker pair, a weighted entropy metric is defined as follows:

$$H = -\sum_{1 \leqslant i \leqslant m+1} p_i \sum_{1 \leqslant j \leqslant n+1} p_{ij} log(p_{ij})$$

where p_i is the percentage of population i out of all cells. Using the gate(s) drawn based on the marker pair, if all target populations are separated from each other and separated from the pre-clustered unlabeled populations, the weighted entropy will be 0. The overlap between target populations and unlabeled populations or split of one target population in multiple gates will increase the entropy, whereas overlap between multiple target populations within the same gate is not penalized. For example, if one marker pair leads to a gate that includes two target populations that overlap, it has equivalent entropy compared to another marker pair that leads to two gates that perfectly separate the two target populations. Therefore, this entropy-based metric tends to gate out the unlabeled populations before separating the target populations. When two marker pairs have the same entropy-based measure, C2G will choose the one with the larger number of gates. In the example shown in Figure 2.2g, the gating strategy in the bottom panel has the lowest entropy and is considered as the best gate and the best marker pair. Compare to the other two marker pairs, the best maker pair excludes most of the unlabeled cells. A small number of cells from unlabeled population 3 (colored green) are carried to next iteration as shown in Figure 2.2i.

2.2.5 Construct a Gating Hierarchy Recursively

We recursively construct a gating hierarchy for the target populations. Starting from the root node which contains all cells in the data, we derive gates on all possible marker pairs and compute the weighted entropy. We identify the best marker pair with the lowest weighted entropy and select the corresponding gate(s) to move forward. The selected gates

form new children nodes. The algorithm iterates to examine cells that belong to each new node, and identify the best marker pair and gate(s) to further separate the target populations and exclude the pre-clustered unlabeled populations (Figure 2.2h and 2.2i). The algorithm stops generating gates for a node when one of the following two conditions is satisfied: (1) the node contains only one population, or (2) for all marker pairs, the automatically generated gate is a trivial gate that includes all cells in the node. Overall, the iterations stop until no marker pair in any node can further separate any target populations. The output of C2G is a tree structure representing the gating hierarchy, and two-dimensional plots of the selected gates at each tree node (except the leaves), as shown in Figure 2.2j. Given the weighted entropy for selecting the best marker pair, this iterative algorithm tends to first exclude pre-clustered unlabeled populations, and then separate the target populations.

2.3 Result

2.3.1 Generate gating hierarchies for populations defined by various methods

To evaluate the performance of C2G, we applied it to generate gating hierarchies for target populations defined in various ways, including manual gating, automated clustering algorithms such as k-means [49, 53] and graph partitioning methods [51], as well as visualization-based methods such as tSNE and SPADE [22, 11]. To measure how well our gating hierarchy captures the target populations, we computed the F-scores [19] for each target population. In addition, we also computed an overall normalized mutual information (NMI), which is widely used for measuring the similarity of graph partitioning results [54]. In this section, we used a published CyTOF dataset on T cells [47] as a testbed, and focused on the 12 surface markers to evaluate the ability of C2G to generate gates for T cell subpopulations. We preprocessed this CyTOF dataset by the inverse hyperbolic sine function with a cofactor of 5.

Table 2.1: **Result on manual gated populations**

Target Population	Node	True Positive	False Positive	False Negative	F-score	Percentage
CD4+ Effmen	7	216	13	8	0.954	2.3%
CD4+ nave	8	2267	30	78	0.977	24.0%
CD8+ nave	6	488	46	37	0.922	5.4%

Target Populations Defined by Manual Gating

In the T cell CyTOF dataset, three manually gated populations were defined by Krishnaswamy et al [47], namely CD4+ effector memory T cells, CD4+ Nave T cells, and CD8+ Nave T cells. We treated these three manually gated populations as target populations and the remaining cells as unlabeled. When applied to this dataset, C2G was able to generate the gating hierarchy in Figure 2.3, where each leaf of the hierarchy in the last panel corresponds to one of the target populations. As shown in Table 2.1, all three target populations were accurately gated with F-scores 0.954, 0.977, and 0.922. The NMI score was 0.893. The percentage of each target population in the dataset is listed in the last column of Table 1, where we can see that the C2G performed well on both abundant target populations and relatively small target populations. Moreover, the gating hierarchy generated by C2G used a similar set of markers as the manual gating analysis that defined those target populations, which is available on page 12 of the supplementary materials of [47]. Compare to the manual gating, C2G did not use CD8 because this dataset has very few CD4+CD8+ events, and therefore, only one of the markers CD4 and CD8 is necessary to gate for the target populations. This example shows that C2G is able to identify a subset of markers necessary for gating the target populations. This feature of C2G enables it to perform panel refinement. We will demonstrate the application of C2G on panel refinement later.

Target Populations Defined by Automated Methods

Many popular clustering algorithms for cytometry data used variations of k-means [49, 53], which motivated us to use clusters defined by k-means as target populations to examine the performance of C2G. We clustered this T cell CyTOF dataset using k-means with k=10 and

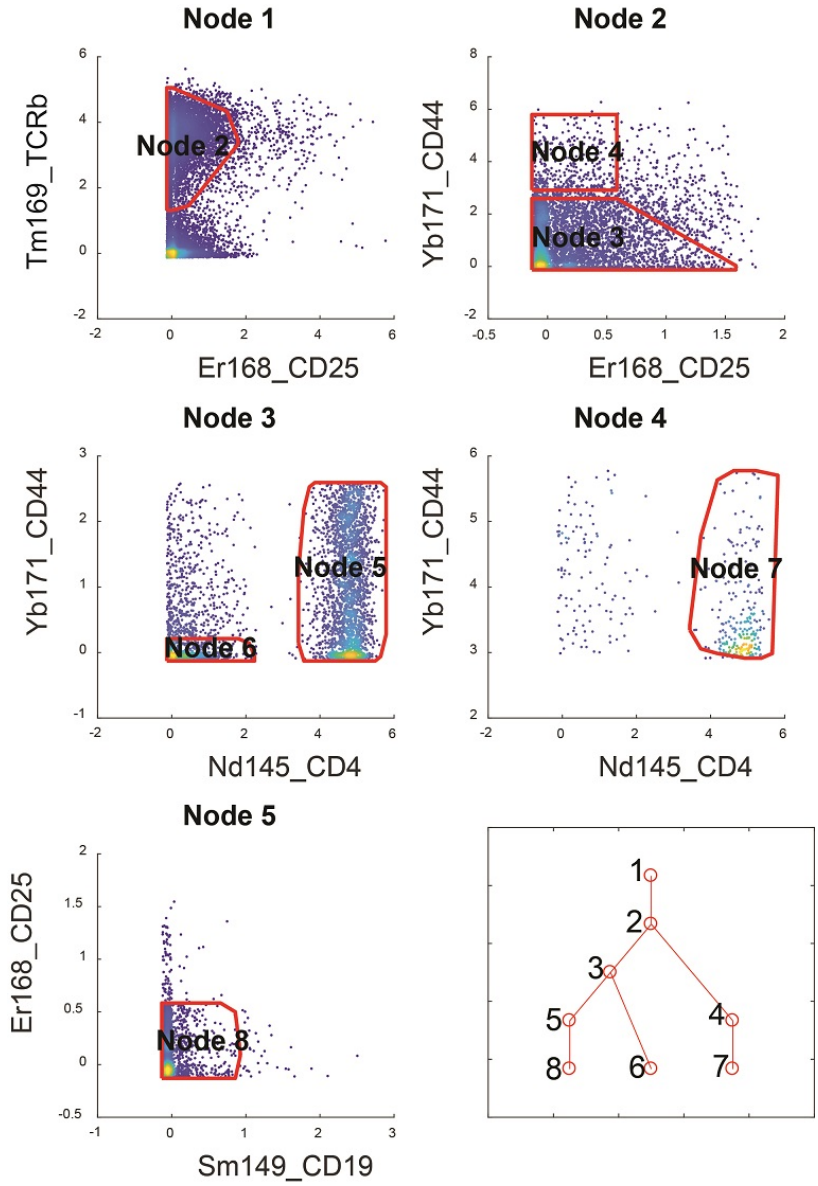

Figure 2.3: **Generate gating hierarchy for manually gated target populations** First 5 subfigures visualize automatically generated gates in 2D biaxial plots. Cells shown in each subfigure belong to the gate indicated by the subfigure title. Each polygon represents a gate. Node 1 is a dummy gate that contains all the cells, meaning that the top-left panel shows all the cells. Each subsequent subfigure shows cells defined by a previous gate. The last subfigure is the tree structure representing the C2G-generated gating hierarchy. Each node corresponds to one gate. The leaves represent the final gated populations generated by C2G, each of which corresponds to one target population.

assumed all 10 clusters are target populations. C2G generated the gating hierarchy shown in Figure 2.4. The gating hierarchy achieved an average F-score of 0.898 for the 10 clusters (Table 2.2) and an overall NMI of 0.770, meaning that C2G was able to generate gates to accurately gate for the k-means defined populations. C2G achieved similar performance on clusters defined by other k-means-like methods including flowMeans and flowSOM. For the same dataset, flowMeans generated 13 clusters, on which C2G achieved an average F-score of 0.884 and NMI of 0.888. flowSOM generated 9 clusters, on which C2G achieved an average F-score of 0.907 and NMI of 0.799.

Table 2.2: **Result on K-means defined populations**

Target Population	Node	True Positive	False Positive	False Negative	F-score	Percentage
1	23	749	145	39	0.891	8.1%
2	9	818	179	95	0.857	9.4%
3	26	1230	275	59	0.880	13.2%
4	10	1900	371	230	0.863	21.8%
5	25	1418	249	74	0.898	15.3%
6	11	1180	200	191	0.858	14.0%
7	18	714	27	38	0.956	7.7%
8	17	138	12	18	0.902	1.6%
9	19	609	38	46	0.935	6.7%
10	20	213	23	5	0.938	2.2%

We also defined target populations using PhenoGraph, which is a nearest neighbor graph-based algorithm [51]. We ran PhenoGraph on this CyTOF dataset and obtained 12 populations. We assumed all the 12 PhenoGraph-defined populations are target populations and applied C2G to derive gates for the target populations. The resulting gating hierarchy achieved an average F-score of 0.707 and NMI of 0.669, which was relatively low. This was because C2G obtained a gate that included populations 1 and 6, but did not generate further gates to separate them. Similarly, C2G gated populations 2, 3, and 8 together (Table 2.3). Upon further inspection, we found that C2G did not separate those target populations because they share substantial overlap with each other in two-dimensional spaces defined by all marker pairs. Therefore, even if those populations may be separated in high-dimensional space, they are not separable by nested two-dimensional gates.

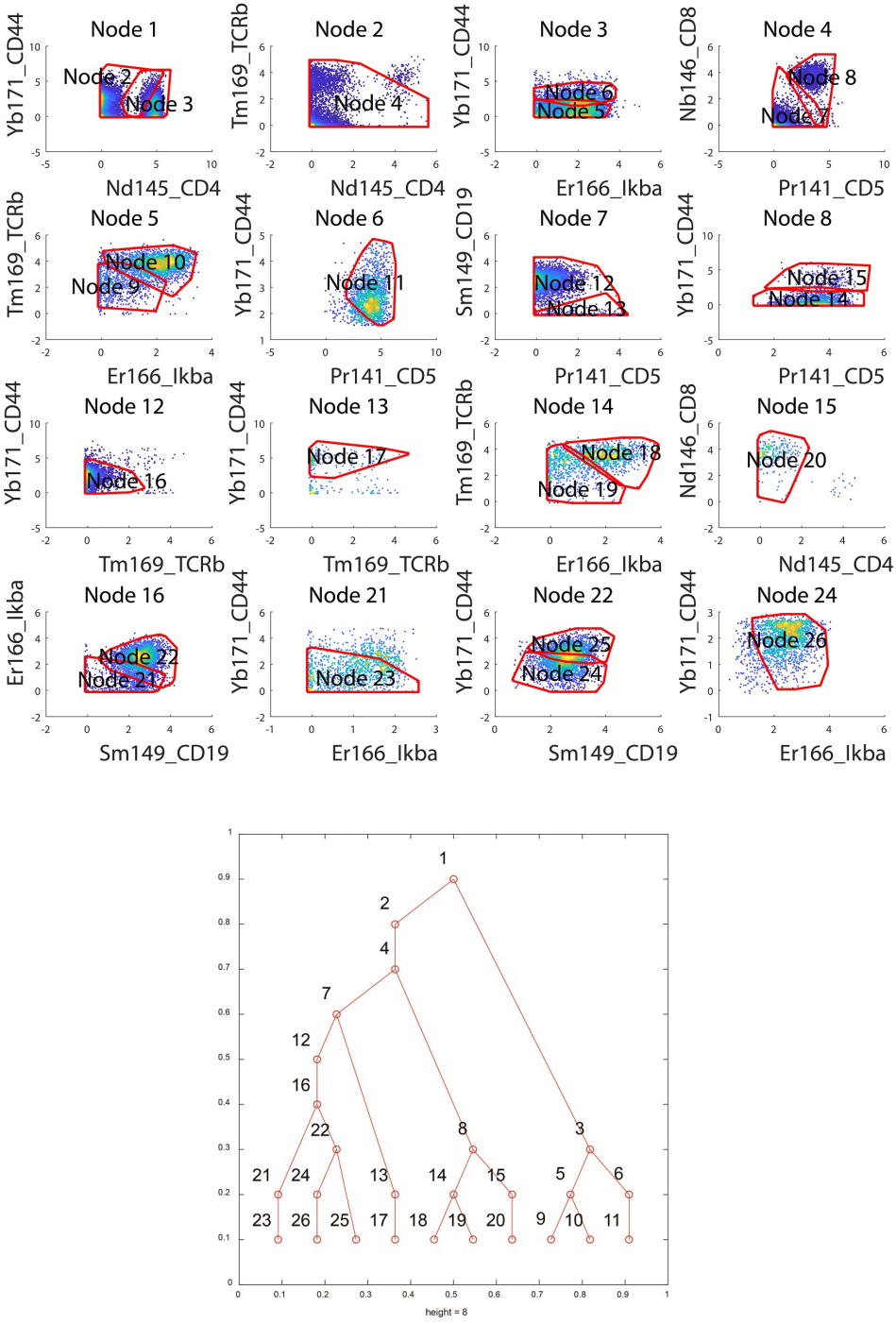

Figure 2.4: **Generate gating hierarchy for target populations defined by k-means clustering** (a) Automatically generated gates in 2D biaxial plots. (b) Tree structure representing the C2G generated gating hierarchy. The 10 leaves represent the terminal gates generated by C2G, each corresponds to one k-means defined target population.

Target Populations Defined by Visualization-based Methods

SPADE and tSNE are methods that can visualize high-dimensional cytometry data in a two-dimensional space, where gates can be drawn to define populations. C2G is helpful for assisting the interpretation of gates drawn on SPADE or tSNE visualizations, by generating a nested gating hierarchy for those gated target populations. We applied SPADE to the T cell data and manually gated the SPADE tree to obtain 4 distinct target populations shown in Figure 2.5a. We then applied C2G to generate a gating hierarchy for the target populations shown in Figures 2.5b, and C2G achieved an average F-score of 0.931 and NMI of 0.907 (Table 2.4).

2.3.2 Challenges in generating automated gating hierarchy

Depending on the data and definition of target populations, automated generation of gating hierarchy may encounter several challenges, such as unlabeled cells, over-clustered pop-

Table 2.3: **Result on PhenoGraph defined populations**

Target Population	Node	True Positive	False Positive	False Negative	F-score	Percentage
1	15	1729	1061	95	0.749	18.7%
2	12	1256	2248	65	0.52	13.5%
3	12	1205	2299	76	0.504	13.1%
4	17	1091	59	72	0.943	11.9%
5	14	881	435	52	0.783	9.6%
6	15	767	2023	34	0.427	8.2%
7	13	748	269	50	0.824	8.2%
8	12	714	2790	34	0.336	7.7%
9	20	252	81	34	0.814	2.9%
10	7	237	46	29	0.863	2.7%
11	19	177	46	15	0.853	2.0%
12	16	140	33	11	0.864	1.5%

Table 2.4: **Result on SPADE defined populations**

Target Population	Node	True Positive	False Positive	False Negative	F-score	Percentage
1	4	76	31	13	0.867	0.9%
2	9	3493	25	63	0.986	36.4%
3	8	1589	23	40	0.978	16.7%
4	6	4334	0	156	0.980	46.0%

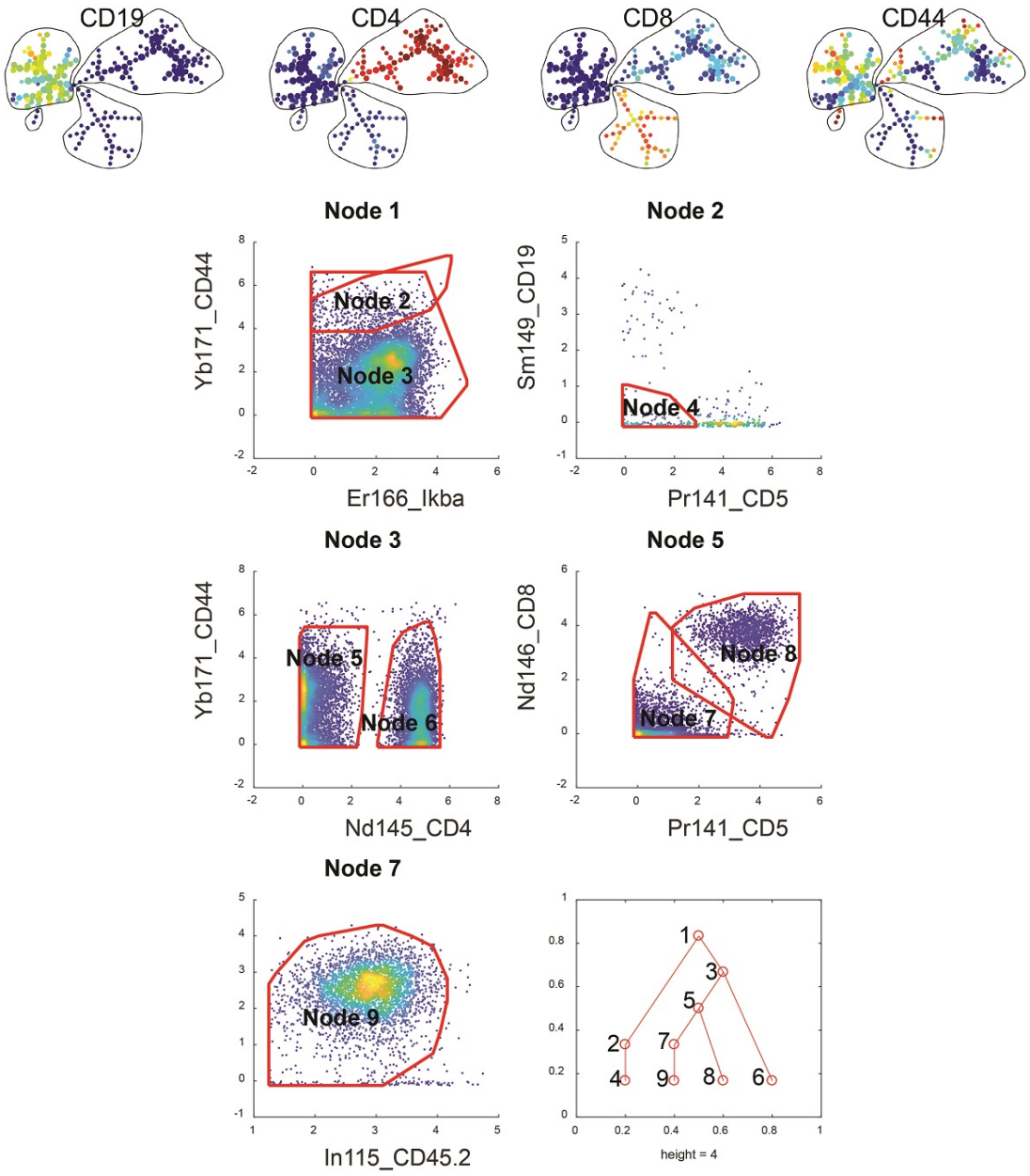

Figure 2.5: **Generate gating hierarchy for target populations defined by SPADE** (a) SPADE tree of the T cell CyTOF data. Four target populations are manually gated on the SPADE tree. These populations represent: B cells, CD8+CD4- T cells, CD8-CD4+ T cells and CD19-CD8-CD4-CD44+ cells. (b) First 5 panels are automatically generated gates in 2D biaxial plots. Last panel is the tree structure showing the C2G-generated gating hierarchy for these four target populations defined by SPADE.

ulations, overlapping populations, large data size, and high dimensionality. We examined the performance of C2G with respected to these challenges.

Unlabeled Cells

When the input labels only account for one or a few of the populations in the data, most of the cells are unlabeled. The unlabeled cells can create challenges in identifying a gating hierarchy for the labeled target populations because the unlabeled cells contain a mixture of other cell populations, debris, and outliers. To address this challenge, we pre-cluster the unlabeled cells according to their marker expression relative to the target populations. As a result, the unlabeled cells are divided into clusters that typically do not highly overlap with the target populations. Although this pre-processing step is likely to generate more clusters than the number of distinct populations in the unlabeled cells, it breaks down the unlabeled cells in a way that is relatively easy to separate from the target populations. In the first example above, the target populations were three T cell subtypes, whereas all other cell types and outliers were unlabeled. After pre-clustering the unlabeled cells, C2G was able to generate a gating hierarchy that accurately captured the target populations (Figure 2.3).

We further evaluated C2G with respect to the number of target populations by comparing three setting. We clustered the T cell CyTOF data using k-means with k equal to 10 and defined the target populations to be all, or three, or only one of the k-means defined populations. Then, we applied C2G to generate a gating hierarchy for the target populations in each setting. Within each setting, we computed an F-score for each target population. As shown in Figure 2.6, C2G consistently achieved high F-score in all three settings, showing the robustness with respect to the unlabeled cell populations.

Over-Clustered Populations

In many applications, the exact number of distinct cell populations in the data is unknown. One popular strategy is to over-cluster the data to ensure all distinct populations are separated [10, 17, 24], and merge the clusters later. Therefore, we tested C2G against input labels that represent over-clustered populations. We performed k-means to cluster the T

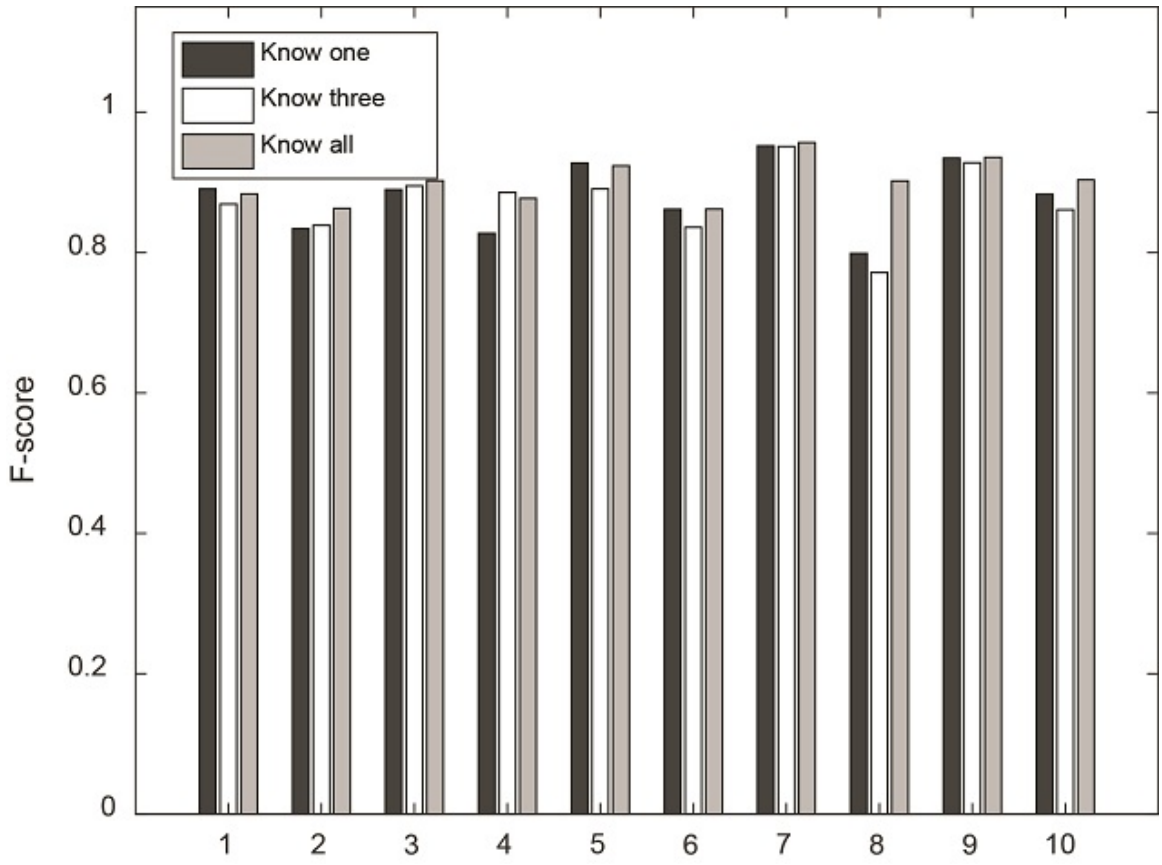

Figure 2.6: **Comparison of cases where one, some or all populations are considered as target populations** Data is clustered by k-means with k=10. Know one means one of the 10 k-means clusters is the target population and the remaining are unlabeled. In each of the Know three setting, three consecutive populations are defined as the target (1-2-3, 2-3-4, , 9-10-1, 10-1-2). Know all means all k-means clusters are target populations. This figure shows that C2G is robust to whether some or all of the populations are considered as target populations

cell CyTOF data and varied K from 3 to 150 and assumed that all k-means clusters are target populations. For each k, we applied C2G to generate a gating hierarchy and computed the average F-score and NMI. As shown in Figure 2.7, the average F-score was above 0.8 up to k=7 and remained above 0.7 even when the number of clusters increased to 150. The decrease of F-score for increasing k was mainly because C2G stopped before attempting to separate certain target populations that cannot be separated in 2D plots. The target populations that cannot be separated in 2D may be biologically distinct populations that can be separated in the higher dimension or over-clustered populations that should be merged.

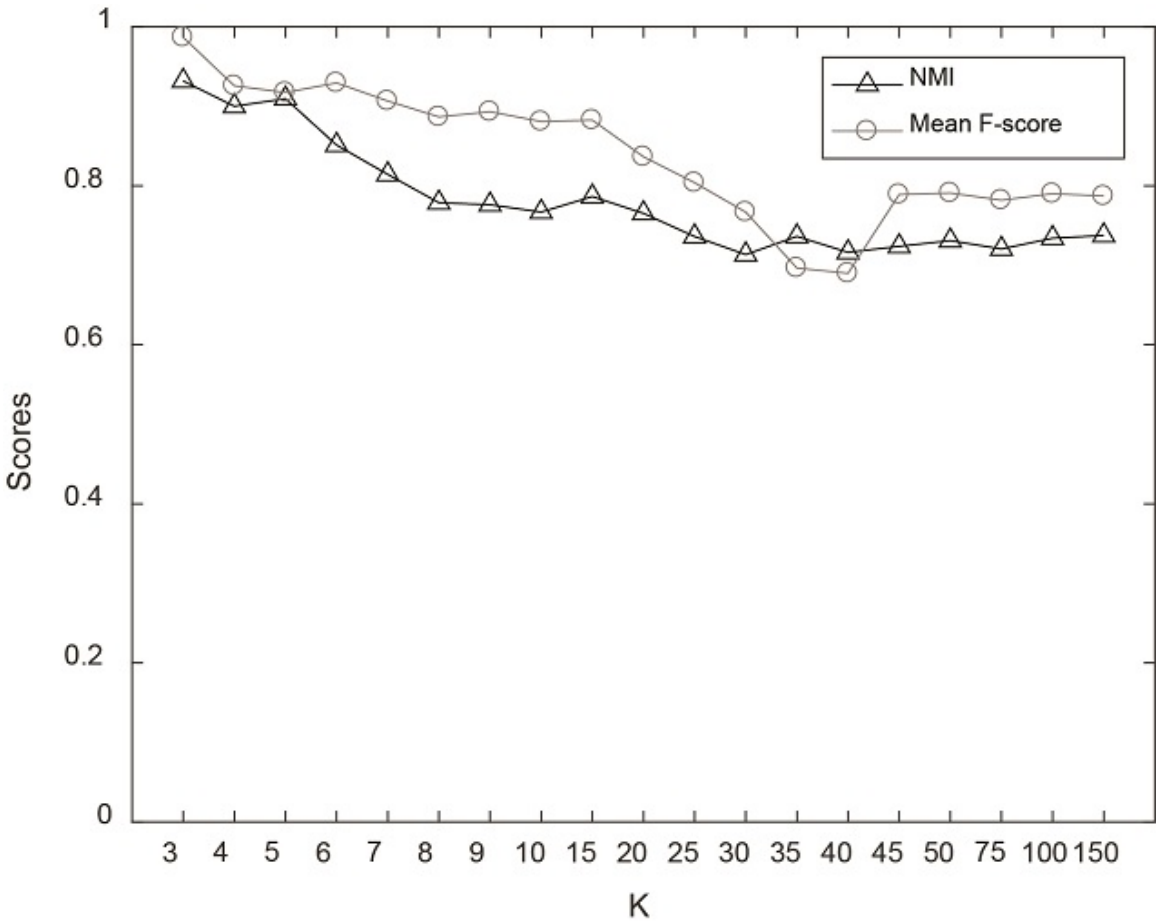

Figure 2.7: **C2G performance with respect to over-clustering** k-means was used to cluster the data with k varying from 3 to 150. For each k, assume all k-means clusters are target populations (Know all) and apply C2G to generate gates. Y-axis represents two scores, the mean F-score and the normalized mutual information. This figure shows that C2G can separate most over-clustered target populations

In addition to varying k in k-means, we also tested C2G on SPADE which intentionally over-clusters the data, and obtained similar results.

Overlapping populations

Populations defined by high-dimensional clustering methods may not always be separable in a gating hierarchy of two-dimensional plots. For target populations that substantially overlap with each other due to over-clustering or high-dimensional marker correlations, C2G is not able to separate them. However, for target populations that slightly overlap due

to measurement noise, we would like to be able to separate them, and we implemented the following in our workflow. Before generating gates based on a marker pair, we first compute the local density of each cell of each population separately. When computing the overlap between populations, we ignore a small percentage of low-density cells. Our default is 5%, and this parameter can be tuned by users based on the data quality and acceptable amount of overlap. When drawing a gate by the union of convex hulls of populations in a cluster derived by MCL, we generate a series of convex hulls by ignoring 0%, 1%, , up to 20% of low density cells, and select the convex hull with the highest F-score as the best gate to move forward. To evaluate this implementation, we artificially added outliers into the manually gated populations above, which induced overlaps among the target populations and the unlabeled populations. The number of outlier cells added to each population was equivalent to a fixed percentage of its size. Applying C2G to the data with various percentages of added outliers up to 32%, the resulting gates always accurately captured the target populations with average F-score above 0.8 and overall NMI above 0.7 (Figure 2.8).

Large Data Size and High Dimensionality

When applied to large cytometry datasets, the running time of C2G is influenced by several factors, including the total number of cells (n), the number of protein markers (d), number of target populations (p), and total number of 2D plots needed to draw gates to separate all target populations. In large cytometry datasets, n is typically in the order of millions. To handle dataset with a large number of cells, we used a grid-based approach in C2G so that most operations are performed on the fixed size grids, and the only time-consuming step in each iteration is the mapping of all cells to the grids, with time complexity of $O(nd^2)$. The total number of iterations varies depending on the dataset and the target populations, and therefore cannot be theoretically estimated.

To test the scalability of C2G, we used a large CyTOF dataset [48] with multiple samples and 21 protein markers measured. We first focused on one sample containing 140,000

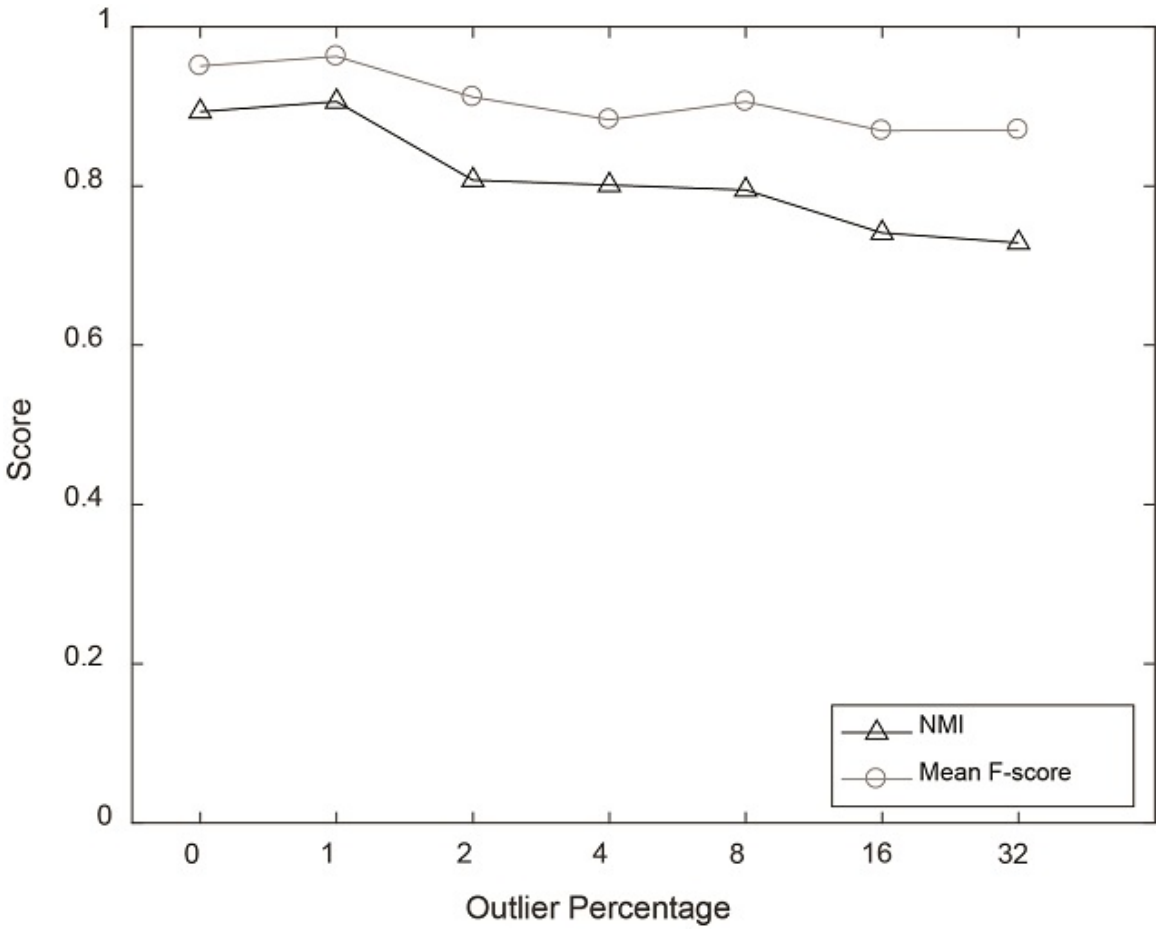

Figure 2.8: **C2G performance with respect to different noise levels that induced overlap among target populations** Various amount of outlier cells were added to the manually gated target populations in the T cell CyTOF data, which induced overlap among the target and unlabeled populations. When C2G is applied to data with the various amount of overlaps, the resulting gating hierarchy accurately captured the target populations

cells, applied k-means with k=10 to define target populations, and applied C2G to identify a gating hierarchy that can gate for these 10 target populations. C2G took roughly 12 minutes to accurately gate for the target populations, achieving an average F-score of 0.867 and an overall NMI of 0.833. We then pooled 9 samples from this CyTOF dataset to obtain an even larger collection of around 1.2 million cells, and performed the same analysis, using k-means to define 10 target populations and C2G to generate gates for the target populations. In this larger example, C2G achieved an average F-score of 0.863 and overall NMI of 0.809, and the running time was around 42 minutes.

2.3.3 Apply C2G to perform panel refinement

As shown above, C2G can serve as a post-processing visualization tool to help interpret high-dimensional clustering results from automated clustering algorithms. In addition, C2G can be used to assist biologists in performing panel refinement, which simplifies the experimental design with fewer protein markers while capturing the same heterogeneity as the original panel.

We demonstrated C2Gs utility for panel refinement using the same dataset as above, the published CyTOF dataset on T cells [47], which contains measurements of 12 protein markers for roughly 9000 cells. We defined cell target populations using k-means based on all protein markers with k=10. When we applied C2G to consider all protein markers to generate gates for these target populations, C2G was able to achieve an average F-score of 0.9, as shown in the left-most point on the curve in Figure 2.9.

We then tested what would happen if the data for one protein marker was unavailable to C2G. In other words, can C2G achieve good F-score with one fewer marker although the target populations were defined using all the markers? For the same target populations, we ran C2G 12 times, each ignoring one of the protein markers. The F-score for gating each target population when dropping the first marker is visualized in Figure 2.10, where we can observe that removal of different markers has a different effect on C2Gs ability in gating different target populations. For example, removal of CD44 significantly impacted the F-scores of most of the target populations, whereas removal of TCRb only affected the F-score of one target population. In order to quantify the relative importance of the protein markers, we use the average of F-scores as a metric. The 12 resulting average F-scores are listed in the bottom row of Figure 2.10, and also visualized in the left-most column of dots in Figure 2.9, where we can observe that ignoring different protein markers led to a different level of decrease in the average F-score. The highest average F-score corresponded to the least important protein marker, which was CD45.1 in this dataset.

After removing CD45.1, we performed the same analysis again, running C2G 11 times

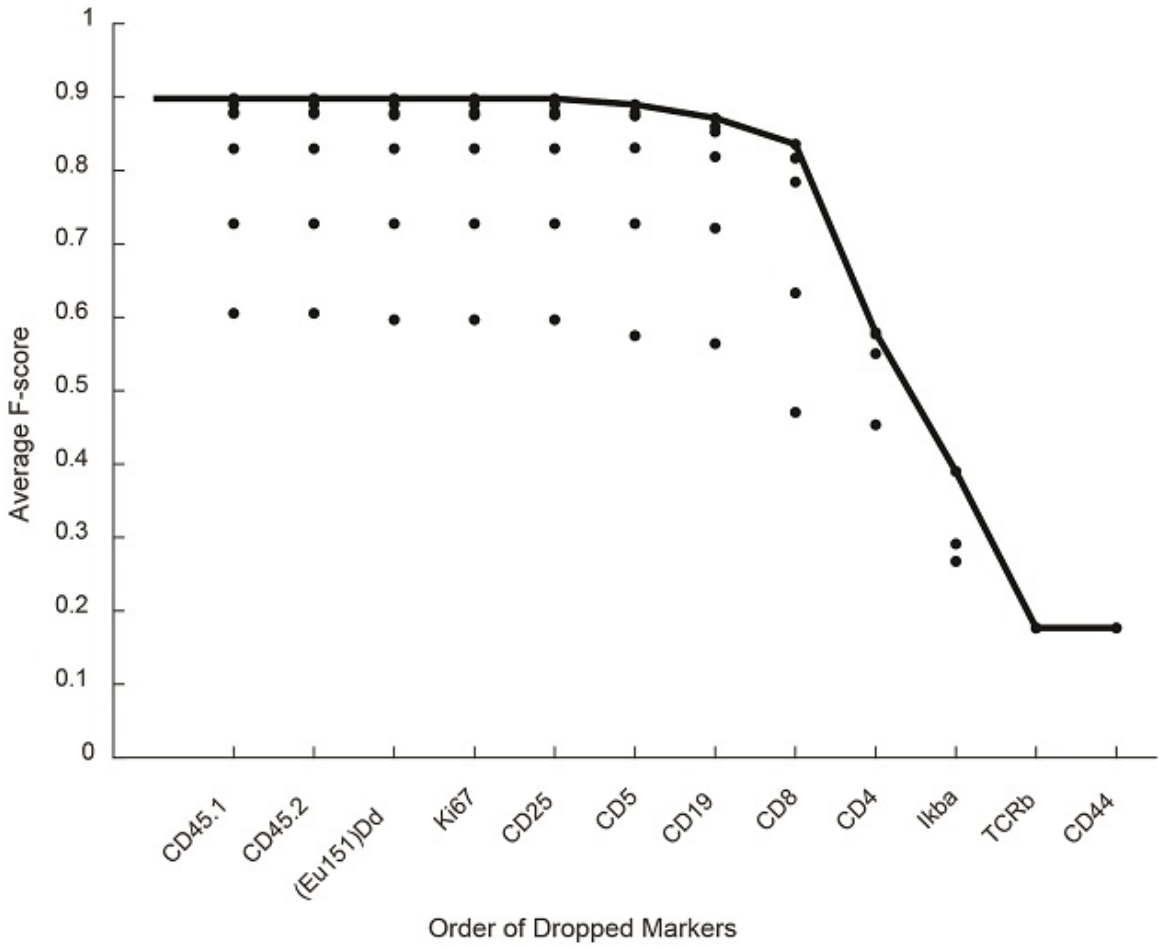

Figure 2.9: **C2G for panel refinement** This figure shows the average F-scores achieved by applying C2G on a subset of markers to gate for target populations defined by k-means on all the protein markers. From left to right, we gradually remove the least informative protein markers shown along the x-axis. The black curve on the top shows the average F-scores achieved by removing the least important marker at each step. The other black dot shows the average F-scores achieved by removing other protein markers

to determine which marker is the least important in the remaining 11 markers. We repeated this process until all markers are removed, and visualized the decrease of the average F-score in Figure 9. The ordering of the removed markers, along with the average F-score, reflected the relative importance of protein markers in this dataset. In Figure 2.9, the highest average F-scores remained the same when we ignored the first five markers (CD45.1, CD45.2, Eu151Dd, Ki67, CD25), indicating these markers are the least useful in this marker panel. The highest average F-scores began to rapidly decrease when marker

Cluster	Perc	F-score	F-score after removing protein marker											
			Cd45.1	CD45.2	(Eu151)Dd	Ki67	CD25	CD5	CD19	CD8	CD4	Ikba	TCRb	CD44
1	8%	0.88	0.89	0.89	0.89	0.89	0.89	0.90	0.92	0.87	0.88	0.43	0.88	0.78
2	9%	0.86	0.86	0.86	0.86	0.86	0.86	0.86	0.86	0.86	0.83	0.48	0.47	0.34
3	13%	0.90	0.88	0.88	0.88	0.88	0.88	0.89	0.86	0.90	0.88	0.63	0.88	0.60
4	22%	0.88	0.86	0.86	0.86	0.86	0.86	0.86	0.86	0.86	0.86	0.85	0.83	0.65
5	15%	0.92	0.90	0.90	0.90	0.90	0.90	0.89	0.86	0.90	0.91	0.91	0.90	0.65
6	14%	0.86	0.86	0.86	0.86	0.86	0.86	0.80	0.86	0.86	0.82	0.83	0.86	0.47
7	8%	0.96	0.96	0.96	0.96	0.96	0.96	0.96	0.96	0.91	0.94	0.69	0.88	0.70
8	2%	0.90	0.90	0.90	0.90	0.90	0.90	0.86	0.73	0.84	0.87	0.90	0.89	0.74
9	7%	0.94	0.94	0.94	0.94	0.94	0.94	0.94	0.94	0.88	0.93	0.62	0.79	0.88
10	2%	0.90	0.94	0.94	0.94	0.94	0.94	0.94	0.94	0.91	0.90	0.94	0.93	0.25
Average		0.90	0.90	0.90	0.90	0.90	0.90	0.89	0.88	0.88	0.88	0.73	0.83	0.61

Figure 2.10: **F-score for each population by dropping one protein marker** F-score of each cell population after removing each protein markers. The protein markers are sorted by the order they are dropped. The result shows that if the CD44 is dropped, most cell populations will result in a bad f-score except for cell population 9. According to the table above, removing Ikba make cell population 9 into a bad f-score, which means cell population 9 is defined mainly by Ikba.

CD5, CD19, and CD8 were removed. In Figure 2.11, we visualize the decrease of F-scores of individual populations along the process of removing protein markers, which showed the same trend as the average F-score in Figure 2.9.

The results show that, although the target populations were defined using all the 12 protein markers, C2G was able to gate them effectively without the first 5 6 six markers. Even if a reduced panel without these markers is used in subsequent experiments, we are still able to more-or-less gate for the populations defined by clustering analysis of all 12 markers in the original panel. To further demonstrate the applicability of C2G on panel refinement, we have performed the same analysis on a larger CyTOF dataset with 140,000 cells and observed similar results. As shown in Figure 2.12, out of the 21 protein markers, removing 13 of them has almost no impact of F-score obtained by C2G.

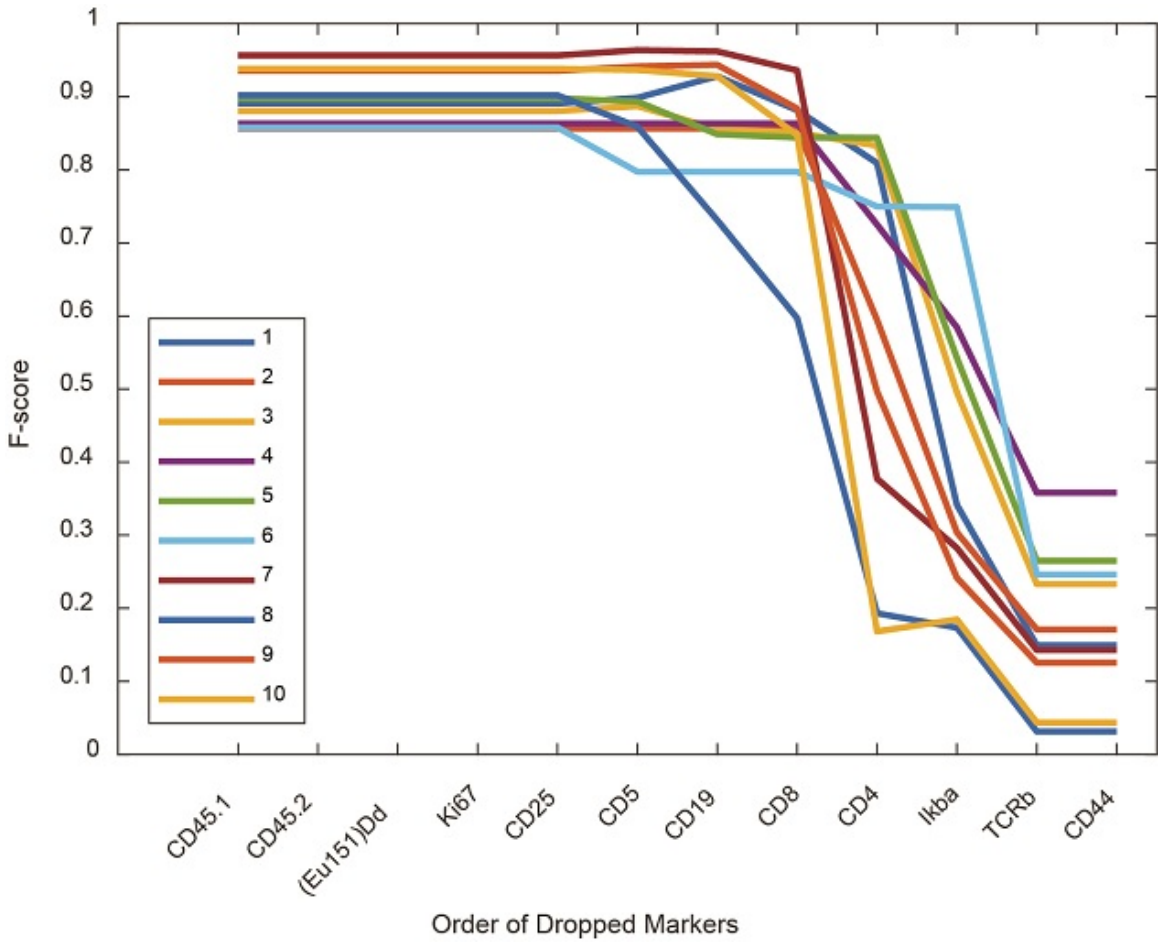

Figure 2.11: **F-scores of all population changes when sequentially dropping markers** Change of F-score of each cell population when dropping the protein makers sequentially

2.4 Discussion

In this work, I provide a new method C2G, which can generate a gating hierarchy of nested two-dimensional gates from populations defined by either manual gating analysis, automated clustering algorithms, or visualization-based methods. C2G fills the gap between high-dimensional analysis algorithms for cytometry data and the conventional gating visualization that biologists are accustomed to, which will help to promote the adoption of high-dimensional analysis algorithms. In addition, the hierarchical representation can also benefit algorithm developers, assisting them to visualize their clustering results for debugging and parameter tuning.

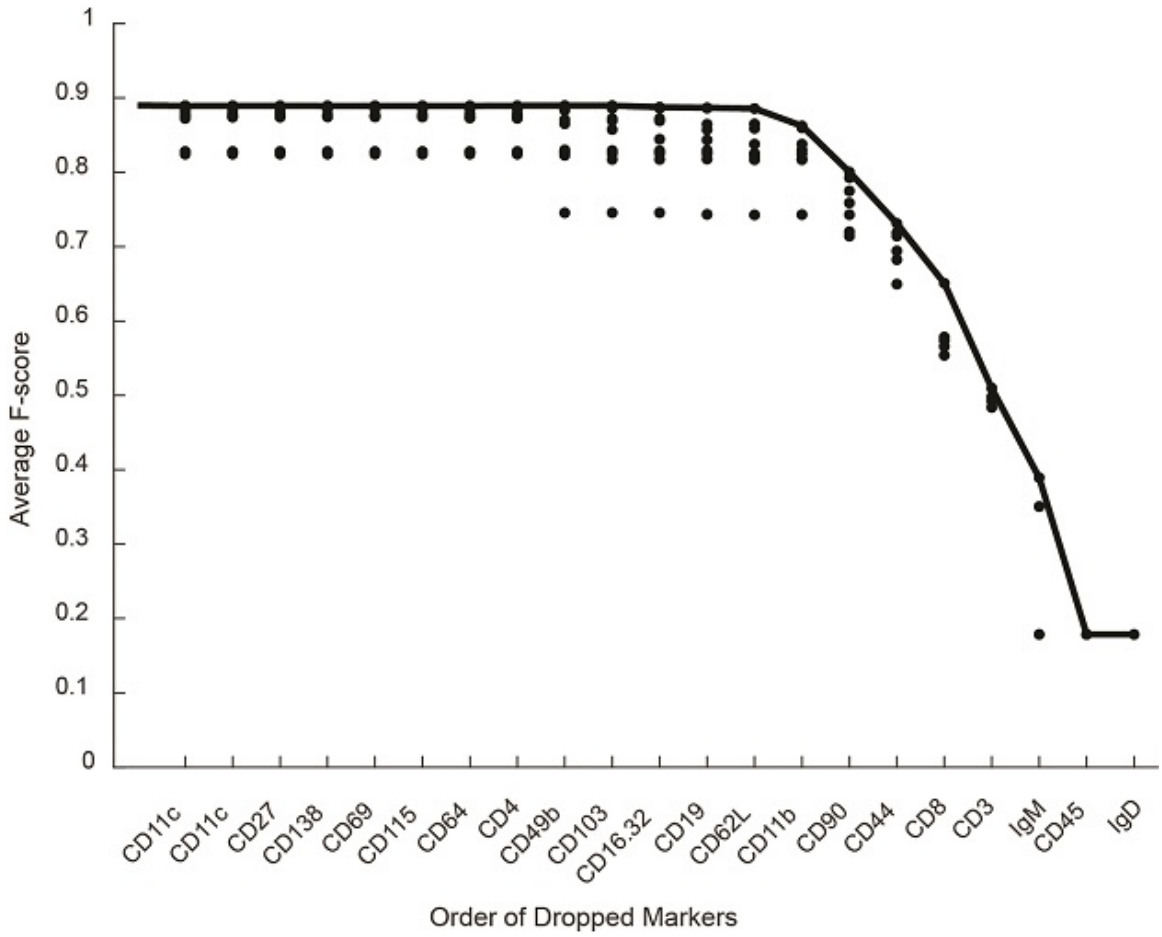

Figure 2.12: **C2G for panel refinement on larger dataset**

Another potential application of C2G is to derive a gating hierarchy for completely unlabeled data. Upon using any general clustering methods (for example k-means or its variants), we can apply C2G to obtain a gating hierarchy to explain the relationship among the clusters. Such a hierarchical explanation is typically not available in high-dimensional clustering algorithms, except for RchyOptimyx [24]. An advantage of this application is that it is flexible and easy to interpret because the gating hierarchy reveals which marker pairs are important in separating which subsets of populations.

C2G can also be used to perform panel refinement. The average F-score from C2G by ignoring one of the protein markers can serve as a measurement of the relative importance of the protein marker. By gradually removing the least informative markers, C2G may reveal that only a subset of the protein markers in the staining panel can effectively explain

cell clusters defined by all the protein markers in the entire panel.

Performance of C2G, of course, depends on the quality of the input, in terms of the separation among target populations and unlabeled cells. For cases where the target and unlabeled populations do not overlap or moderately overlap, C2G is able to successfully generate gating hierarchies to gate for the target populations. As to cases where the target and unlabeled populations substantially overlap in all 2D spaces, C2G is not able to separate them. However, this provides an indication of potential problems with the clustering analysis that defined the target populations, which can serve as diagnosis and quality control for the clustering analysis.

CHAPTER 3
KNOWLEDGE-DRIVEN METHOD TO AUTOMATICALLY ANALYZE FLOW/MASS CYTOMETRY DATA

3.1 Introduction

Flow cytometry and CyTOF are powerful technologies that provide multi-parametric single-cell data of heterogeneous populations of cells [8, 9]. The data for one biological sample is usually in the form of a tall thin matrix, where each row corresponds to an individual cell and each column corresponds to one protein marker. Each element in the data matrix is the expression of a protein marker in/on an individual cell. The typical number of cells is larger than 100,000. The typical number of protein markers is up to 12 for flow cytometry [10], and up to 50+ for CyTOF [9]. Such single-cell data enables quantification of cellular heterogeneity [11, 12], discovery of novel subpopulation [13, 14] , identification of rare cell types [15], and correlation between single-cell characteristics and clinical features [16, 17].

Analysis of flow cytometry data often aims to identify cell populations with distinct phenotypes, which is essentially a clustering problem. The traditional manual gating approach produces visualizations that are easy to interpret but is subjective and labor intensive. Motivated by the drawbacks of manual gating, great efforts have been spent to develop automated data-driven algorithms. Those data-driven algorithms operate in an objective manner, but usually do not take advantage of any existing knowledge and suffer from difficulties including rare populations, large data size, high dimension, and indirect interpret-ability, etc., [19, 20]. To overcome the above difficulties, we propose to analyze the cytometry data in a knowledge-assisted way. However, for different marker settings, the prior knowledge required to analyze the cytometry is very different. To automatically

select the right knowledge for different marker settings, we need to build a knowledge graph that standardizes knowledge from different sources. In this chapter, I will represent a framework that can learn knowledge from existing manually gated flow/mass cytometry and automatically generate gating strategy for new data with different experiment settings.

3.2 Method and Result

3.2.1 Data Collection for building knowledge-graph

Existing gated cytometry data can be good materials to build the knowledge-graph. Sources of manually gated cytometry data are available in platforms like Cytobank and Flowrepository [55, 56]. Some experiments in Cytobank contain acs files that might provide descriptions of gates [52]. Some experiments in Flowrepository provide the source file of FlowJo that contain the gating information, which can be interpreted by flowWorkspace [57]. However, flowWorkspace only supports XML files generated by FlowJo 9.2 or earlier and only a small portion of existing data provide such format. Another difficulty in integrating cytometry data from different sources is that cytometry data generated by different methods or from different materials might have different cell component. Therefore, cell populations from different data sets are difficult to be matched.

However, for the purpose of building the proposed knowledge-graph, we are only interested in the order of marker usage and binary marker expression (positive/negative) of each cell population. Getting the expression level for each individual cell is time consuming and not necessary at this stage. To quickly collect a large amount of manually analyzed data, I use Google image search to find images that match the following keywords: flow cytometry quadrant gate, gate lymphocyte flow cytometry. With the browser simulator based on selenium(a python package), I am able to automatically download the top 1000 images for each keyword. Then, I manually remove images that are not manual gating results and selected 100 unique and high-resolution images for the following experiment.

3.2.2 Knowledge Graph Definition

To develop a representation that integrates existing knowledge of gating and clustering analysis, I propose to use directed graphs. In a knowledge-graph, each node represents a cell type. Each node receives a biologically meaningful name (e.g. natural helper lymphocyte) and contains information of its surface marker combination (e.g. CD3+ CD8+ CD4-). A directed edge AB means that cell type B is a sub-population within cell type A, and it is possible to draw one gate on cell type A to obtain cell type B. Information attached to each edge is the positive/negative expression of protein marker pairs. In some situation, only one of the marker pair is important in drawing a gate, the other markers are marked as neutral.

3.2.3 Building Knowledge-Graph

According to the definition of knowledge-graph, each manual gating strategy can be viewed as one sub-tree of the knowledge graph. Learning a new gating strategy is to connect the new sub-tree into the knowledge graph. The overall work-flow is as follows: 1. Check if nodes in the new sub-tree are already in the knowledge graph. If not, insert the new node into the knowledge graph. 2. If two nodes are connected in the new sub-tree but their corresponding nodes are not connected knowledge graph, connect them by inserting a new edge. This inserted new edge also contains information about whether a cell type appears negative/positive of a certain protein marker. Unfortunately, this procedure can suffer from multiple issues that require additional strategies to handle. For example, cell types in new data use different name convention from existing cell types in the knowledge graph. I will discuss them in the following sections. In general, one gold standard in handling any potential issue is to fully explore all information delivered in the new data. In other words, all protein marker used in the new dataset is assumed to be useful and contain some of experiment designerś expectation on the outcome of the dataset. For example, if a dataset contains both CD3 and CD19, there is high chance that this is a lymphocyte dataset and

they are interested in either B cell or T cell.

Matching Cell Types Names and Protein Marker Names

In a different dataset, the markers and cell types are named under different conventions. For example, some publications attach dye names to marker name while some not. Some publications use T cell while others use T Lymphocyte. In order to match marker and cell type names from different sources, we need to standardize them. This standardization includes two major parts. First part is to discard grammar/format difference like capital letters, plural, and dye names. The first part is simple and I match them by manual inspection and implementing different rules to handle them one by one. For example, discarding all punctuation, space, and 's'/'es' at end of each word of a name. The second part of standardization is to match different naming conventions. Matching cell type names are done using the cell ontology database [58]. This database contains a total of 2319 cell types in the hierarchical structure. 861 of them contain information about names and synonyms (one cell type can have multiple cell names). After discarding the grammar/format difference, the framework will match cell population from knowledge-graph to cell types in the cell ontology database by name. If the names of two cell populations are matched to the same cell types in the cell ontology database, they are treated as the same node. As a note, the cell ontology database contains 86 redundant cell types that different cell types are named the same in the database, I only keep the one appear the first in the database. This simplification won't hurt the results for the purpose of matching convention name.

However, even the standardization procedure described above can help matching a large amount of cell type names, it might not fully solve the matching problem. Other than the fact that the cell ontology database is not comprehensive, the other issue is that data from different publications might be generated from different experimental designs and conditions. In other words, cell populations with the same name might be different cell populations and should be gated differently. Such an issue cannot be solved solely by name

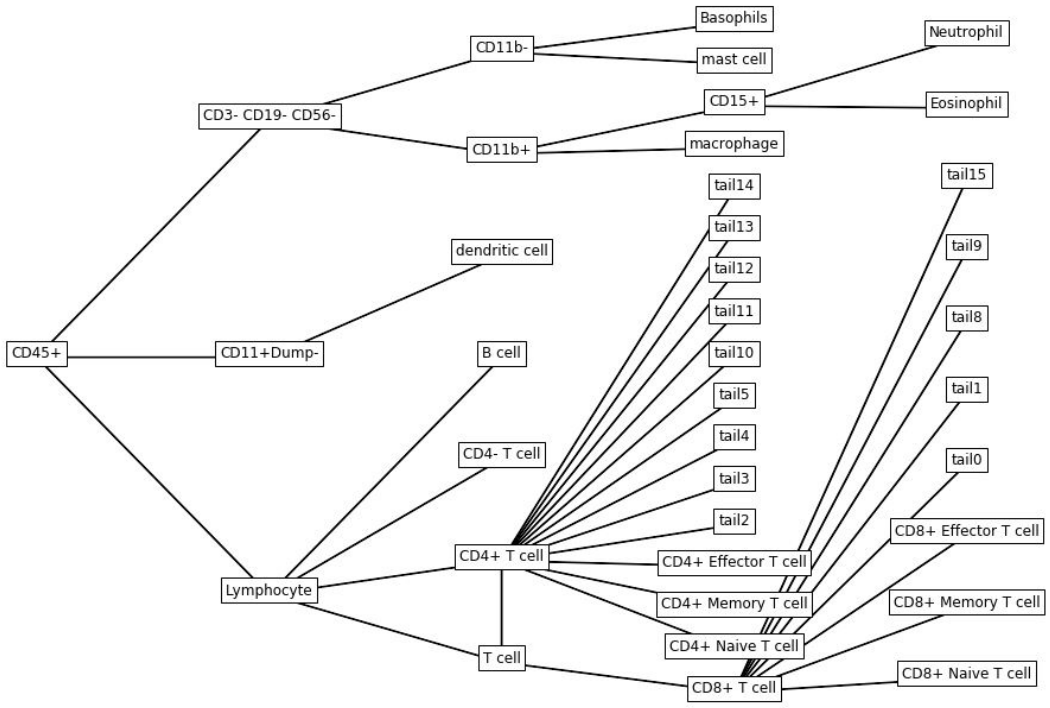

Figure 3.1: **Knowledge-graph built from ten gated data** This is a simple knowledge-graph built from 10 gated flow cytometry data. The nodes named ”tail” followed by a number are placeholder nodes that we do not know their exact name.

matching and require a more sophisticated understanding of the publication of corresponding data source.

Handle Unnamed Cell Types

In some manually gated dataset, the biologists might not give all gated cell populations a name, instead, they simply name them as Q1, Q2, and so on. In addition, we usually do not know the starting point of a gating strategy when just view the image of manual gating. The naming convention and unknown starting point both bring in some unnamed nodes, which cannot be matched to each other or anything in the knowledge graph. However, I do not want to discard such rich information as it is so common and some cell populations are never given a formal name. My strategy is to name each of unnamed node as a distinct placeholder and insert it into the knowledge-graph as shown in Figure 3.1. After

inserting all the placeholders from all data-sets, the knowledge-graph will have lots of unnamed/redundant branches. To simplify the branches, I need to re-compile the knowledge-graph. With the fact that some of the placeholders might be the same cell type, I will check the expression of all protein marker for each of the nodes in the knowledge-graph and merge nodes with exactly the same protein binary expression(positive/negative). When merging two nodes, the edges connected to them are also redirected to the new merged nodes. As a note, each time we have a new manually gated dataset, we need to build the knowledge-graph with all previous data-sets from scratch and compile it.

It is true that such merge can potentially merge two cell populations that are actually different; they have same marker expression in the knowledge-graph just because not enough information to differentiate them from existing gated data. Such a problem can be alleviated by collecting more data. Given a limited amount of data, my current strategy intends to make full use of existing data and not merging these nodes will not make the knowledge-graph more useful.

3.2.4 Application of knowledge-graph

This knowledge-graph has two main applications. One is to automatically suggest gating strategy for new data given marker settings. The other is to suggest a panel design given interested cell types. This second application is trivial since we can simply trace the ancestors of the interested cell types in knowledge-graph and use all the markers along the way as the marker setting. In this section, I will focus on describing how to suggest gating strategy for new dataset.

Given marker setting of a new dataset, the knowledge-graph will find the maximum sub-tree of the knowledge graph that covers the largest number of given markers. To be more specific, given a new dataset, our algorithm will pick up all the edges that are available from the current marker setting. One edge is available if all the positive/negative marker recorded in the knowledge-graph exist in the new dataset. If all the selected edges are

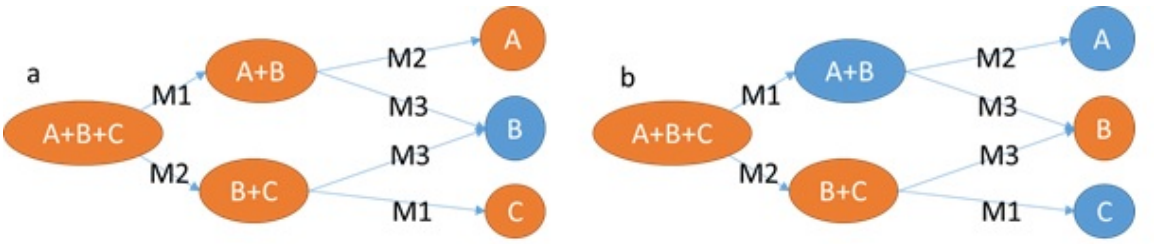

Figure 3.2: **Example of suggesting gating hierarchy based on known marker settings** Above is an example of suggesting gating hierarchy based on known marker settings. Each node in the graph is one cell population and each edge means some publication has drawn such a gate using the corresponding marker. The orange nodes are reachable and blue are not. a) A dataset with marker M1 and M2. We can gate for node A and node C, but not node B. b) A dataset with marker M2 and M3. We can gate for node B.

connected, the connected sub-graph is the final gating hierarchy as shown in Figure 3.2. In this figure, the connected orange nodes are all the reachable nodes under the current marker setting and the sub-tree connecting the reachable nodes will be the final gating hierarchy. However, in the real application, some of the picked edges are not connected to others and multiple sub-graphs will be obtained. The sub-graph that covers the largest number of given makers will be the suggested gating strategy.

3.2.5 Draw Gates on Given Marker Pair

Object Detection Strategy to Identify Cell Populations

Cell populations in flow/mass cytometry are conventionally drawn in 2D scatter plots and gated by their local density. Biologists draw gates around the density peak to define the cell populations. Therefore, viewing the 2D density plots as an image, we can treat the gate drawing problem as an object detection problem. In recent years, the development of deep learning brings breakthroughs to the field of object detection. The state of art strategy in object detection are two branches. One is RCNN [59], which use the semantic profile to predict the potential location of an object and predict class for each location. Another popular strategy is You Only Look Once (YOLO) [60], which predict both location and class in one stage. RCNN performs better when the number of classes grows large

and YOLO is more time efficient and easy to train. In the case of flow/mass cytometry gating, people are usually interested in regular cell populations(density peaks) and rare cell population. Density peaks can be treated as one class and rare cell population can be treated as another class, which in total give us only two classes. Therefore, we do not have a high requirement in classification and decide to use a YOLO-like strategy to automatically detect cell populations in 2D density plots.

My YOLO-like strategy relies on the convolutional neural network which uses sliding-window-like strategy to scan for important features. The input of the neural network is the transformed 2D density plots. The density plot is an image of 50-by-50 pixels. The value of each pixel is computed by counting the number of cell events located in that area. The output of the neural network is an 8-by-8-by-5 tensor. This output clips the original image into 8-by-8 pieces and uses a vector of length 5 to describe whether and where a cell population locates in the grid. In this vector, the first element C is a value between 0 and 1 resulted from sigmoid activation. A value close to 1 means this grid is highly likely to have a center of one cell population located inside. The following two values (x and y) are the relative coordinates of the center in the grid. These two values are linear output from the neural network and range from 0 to 1. The last two values are the size of the object, to be more specific, they are the square root of the width (w) and height (h) of the cell population.

Evaluation and Results

The predicted gates will be evaluated by intersection-over-union (IoU), which is the area of intersection of the bounding box over the area of the union of bounding box between ground truth and prediction. Figure 3.4 is an example of how the IoU is defined. If one 2d plot contains multiple gates in ground truth, the IoU is computed for each gate separately. The IoU for each gate is the maximum IoU between this gate and any of the predicted gates. Out of 3000 simulated 2D plots, 96 percent of gates obtained an IoU over 0.5, which means the

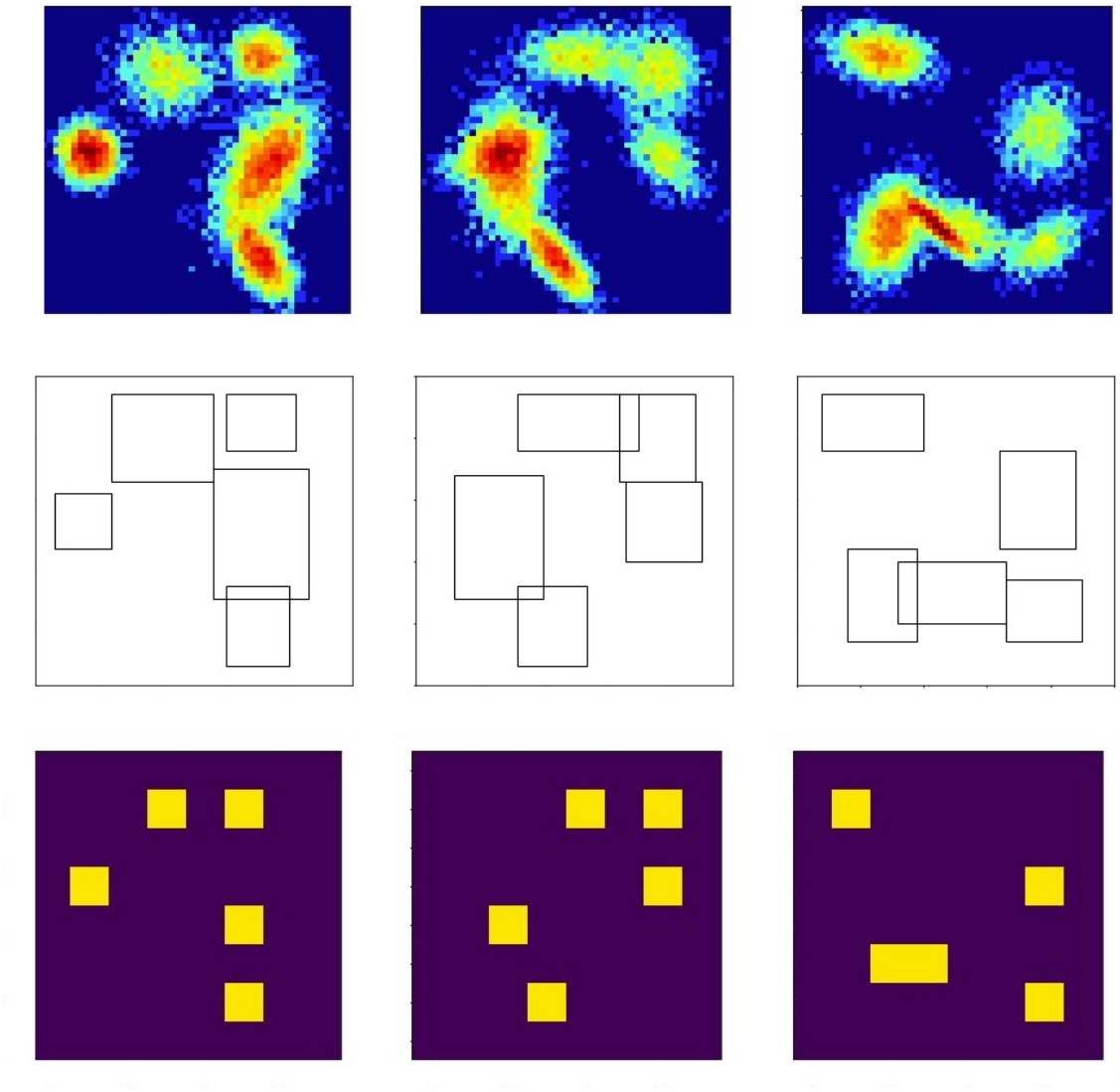

Figure 3.3: **Example of data preparation** In this figure, each column corresponds to one example (one 2D density plots with gated cell population). The first row is the 2D density plot. The color in the in the image represent the local density. Red means a high-density area while blue means a low-density area. The second row shows the bounding box for each cell population. Overlapping between the bounding box is allowed. The third row shows ground truth of the first element of the 8-by-8-by-5 tensor. Yellow rectangle means the center of one cell population exists in the grid while blue means no center of cell population exists in the grid.

neural network successfully captured 96 percent of known populations. However, only 16 percent of gates obtained an IoU over 0.9, which means the neural network cannot perfectly predict the sizes of the bounding boxes. This limitation is acceptable in real application because the exact size of the bounding box usually does not impact the interpretation or

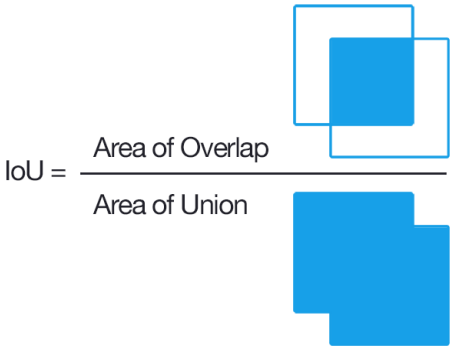

Figure 3.4: **Definition of IoU**
This image is downloaded from
`www.pyimagesearch.com/2016/11/07/intersection-over-union\-iou-for-object-detection/`

annotation of the gates. Figure 3.5 is an example of gates drawn on one real dataset [61].

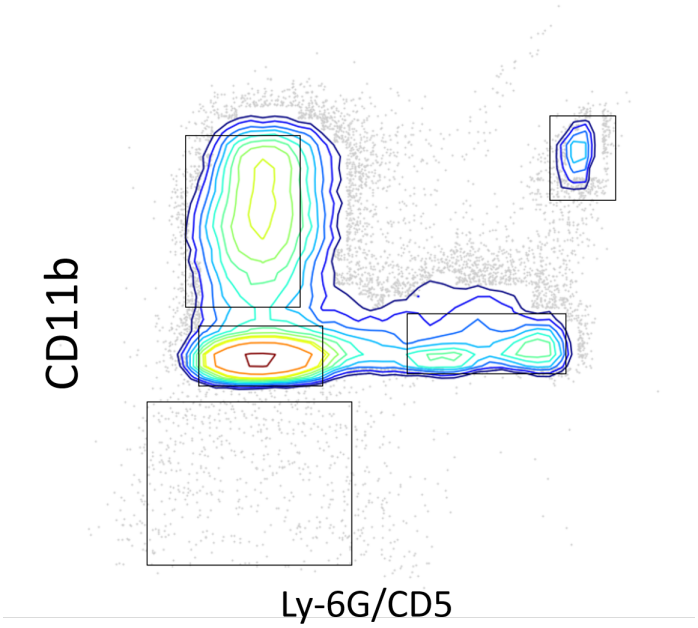

Figure 3.5: **Gates drawn on real data**
This is a density plot of real flow cytometry data. The colored contour represents density of cells. The gray dots are outliers with low local density. The black rectangles are gates predicted by the neural network.

Data Preparation for Drawing the Gates

The images of manual gating results collected in the previous section cannot be directly used to train the neural network because the images already have gates drawn on it and the drawn gates will make the training easily over-fitting. In response to this situation, I have to

collect gated mass cytometry data in raw format (FCS and gatingML) and draw the density plots according to the gates shown in the original papers. This process is slow, and I can only curate a small amount of data. In order to obtain enough amount of data to train the neural network, I combine both data augmentation and simulation to achieve a much larger set of data.

To be more specific, I randomly generate data-sets with multiple cell populations in high-dimension and project them to each of the 2D marker pairs to draw a density plot. By doing this, I also know the gates for all cell populations in the simulated density plots. Upon combining the simulated density plots with real data, I then perform an elastic deformation [62] on all the density plots to generate a distorted version of the data. This augmentation step allows me to have a lot more data for training. As a note, some of the training data only have gates for a subset of cell populations. This situation is handled by designing the cost function, which will be detailed later.

As mentioned earlier, the output of the proposed YOLO-like neural network is an 8-by-8-by-5 tensor. However, existing gates are usually not present in this way. For quadrant gates, two numbers are used to describe the threshold of the gates. For polygon gates, coordinates of the vertex are used. For circles and eclipses, at most six parameters are used to describe one gate. Some researchers also use a mask to describe an arbitrarily shaped gate. To standardize the different ways of defining gates into a format that can be obtained by the YOLO-like neural network, I first summarized each gate into a bounding box. The bounding box is drawn by the 2 and 98 percentile of the corresponding cell population both vertically and horizontally. Each of the bounding boxes is assigned to one of the 8-by-8 grids based on the coordinate of its center. Some examples of how the manual gating is standardized are shown in Figure 3.3. The third row of the figure shows the first element of each grid: each grid colored yellow contains one known cell population. In fact, to preserve the shape of the bounding box, the 8-by-8-by-5 tensor also contains the relative coordinate and size of each bounding box as described above.

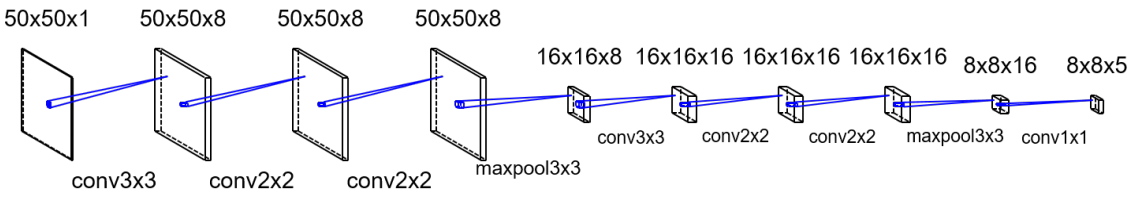

Figure 3.6: **Architecture of Neural Network**

Training

The architecture is shown in Figure 3.6. The neural network has six convolutional layers with ”same” padding and ”relu”[63] activation. Each convolutional layer is followed by a batch-normalization[64] operation. After the third and sixth convolutional layer is max-pooling layers, which enable the neural network to down-sample the 50-by-50 density plots into 8-by-8-by-5 tensor. During the training process, a batch of 64 density plots is stacked and passed together to the neural network until all density plots are used in training for 200 iterations. Adam [65] optimization with a learning rate of 0.001 is used to update the weights in the network.

As mentioned earlier, some training data do not have gate for some cell populations. To correctly learn the gates in this 2D plots, the loss function is defined by

$$
\begin{aligned}
loss =& 2 \sum_{g}^{all\ grids} 1_g^{obj}[(x_g - \hat{x_g})^2 - (y_g - \hat{y_g})^2] \\
&+ 2 \sum_{g}^{all\ grids} 1_g^{obj}[(\sqrt{w_g} - \sqrt{\hat{w_g}})^2 - (\sqrt{h_g} - \sqrt{\hat{h_g}})^2] \\
&+ \sum_{g}^{all\ grids} 1_g^{obj}(C_g - \hat{C_g})^2 \\
&+ 0.2 \sum_{g}^{all\ grids} 1_g^{noobj}(C_g - \hat{C_g})^2
\end{aligned}
$$

This loss function is computed across each of the 8-by-8 grids. 1_g^{obj} means grid g has a cell population in it as ground truth. In this situation, the loss function includes three parts:

mean square error of the relative coordinate (x_g and y_g), mean square error of square root of size of bounding box ($\sqrt{w_g}$ and $\sqrt{h_g}$), and mean square error of confidence prediction (C_g). The first two parts have a weight of 2 and the third part have a weight of 1. 1_g^{noobj} means no cell population is in grid g. When no cell population is in the grid, 1_g^{obj} become zero and only the confidence loss contributes to the final loss and such loss only has a weight of 0.2. Such definition of loss function punishes more when the neural network fails to identify an object that labeled in ground truth. This feature properly solves the problem when only subsets of cell populations are labeled.

3.3 Discussion

The work discussed in this section is a prototype of the knowledge-graph and has some weaknesses. First, the data is collected by Google image search which might not cover all important gating strategies as Google images search only presents images that are relevant to the keywords, not the research topics. Second, we only use the cell ontology database to match cell types. This database is not comprehensive, and many conventional names cannot be correctly matched. To address the first two weaknesses, more human labor should be adapted to collect data and match cell types. The third weakness is that this knowledge-graph cannot handle protein markers that are not in any of the training data (unknown markers). To address this issue, one possible solution is to run the knowledge-driven approach with the known makers to define a set of cell populations and then use data-driven methods to compare the cell populations with the unknown markers. In this chapter, I have also provided a deep learning approach to automatically identify the cell population in 2D scatter plots. The identified cell populations can be matched to knowledge-graph by their binary expression of markers. However, in the real application, not all cell populations can simply be described by binary expression. To overcome this issue, a possible solution is to adapt graph-matching techniques to match the identified cell populations to manually gated cell populations.

CHAPTER 4

RECONSTRUCT LINEAGE TREE OF B CELL RECEPTOR GENE

4.1 Introduction

To specifically recognize and respond to different pathogens, adaptive immune system relies on a diverse repertoire of B cell receptors (BCR). Such a diverse repertoire comes from recombination, somatic hypermutation of immunoglobulin (Ig) gene segments, and selection by their ability to bind pathogens. This process will generate numerous different BCRs. To explore the dynamic process of BCR affinity maturation, researchers have applied high throughput sequencing [30, 31, 32] to examine BCR repertoires and to construct lineage trees of BCR sequences [33, 34].

In a BCR lineage tree, each tree node corresponds to one unique sequence, and each directed edge indicates the relationship between one sequence and its immediate ancestor, which are separated by one-base mutation, insertion or deletion. Given high throughput BCR sequencing data of a repertoire, the observed sequences correspond to some of the internal nodes and the leaf nodes of the BCR lineage tree, while many intermediate nodes are not observed due to the diversification and selection process the repertoire went through, as well as subsampling inherent to the assay. With these observed sequences, we can easily identify the root sequence of this tree by sequence alignment against known germline BCR segments in the genome [35]. To reconstruct the full lineage tree, we need to fill in the unobserved internal nodes and connect them to the observed nodes by direct edges. This process is similar to building the phylogenetic tree among species, except that only leaf nodes are observed in the phylogenetic problem, whereas some internal nodes and the root nodes are also observed in this BCR lineage tree reconstruction problem. Popular methods in the phylogenetic analysis include maximum parsimony, maximum likelihood, and

Bayesian methods. The maximum likelihood and Bayesian methods are usually computational demanding and require a decent amount of prior knowledge [36, 37], for example, the replacement rates and preference of mutation target under selection pressure [38]. Such prior knowledge is relatively limited in BCR lineage trees. Therefore, we decided to pursue the maximum parsimony idea to reconstruct the BCR lineage tree, which intends to reconstruct a tree as small as possible, which connects all the observed sequences with the minimum number of mutation, insertion and deletion events.

Reconstruction of a phylogenetic tree using the maximum parsimony method is known to be an NP-Complete problem [39, 40], which means no guarantee of the best solution in polynomial time. In terms of computational complexity, reconstruction of BCR lineage tree is very similar to a phylogenetic tree. Although the root sequence and some of the internal nodes are known, it is still an NP-Complete problem [34]. To reconstruct BCR lineage trees from high throughput sequencing data, an algorithm named IgTree was previously developed, which is a heuristic procedure consisting of multiple components [34]. It first constructs a preliminary tree that only contains observed sequences based on multiple sequence alignment [41], then uses a complex scoring metric to gradually add internal nodes to complete the full tree, and finally scans the resulting tree to identify subtrees that can be reduced by reversion events. IgTree enabled efficient analysis of large BCR sequence datasets, and brought insights into various area of somatic-hypermutation-related biological processes including neutralization of HIV antibodies [42] and progression of follicular lymphoma [43]. Another algorithm for reconstructing BCR lineage trees is included in the TIgGER software package [44], which is a classical maximum parsimony method called ”dnapars” in the PHYLIP library [45, 46]. dnapars almost always achieves smaller lineage trees compared to IgTree but is considerably slower when analyzing a large number of observed sequences.

Here, we present a novel algorithm, GLaMST, to reconstruct BCR lineage trees using the maximum parsimony criterion. GLaMST uses simple heuristics to Grow the Lineage

trees along the Minimum Spanning Tree. In terms of the maximum parsimony criterion, GLaMST generates lineage trees with a small size similar to dnapars and outperforms dnapars for simulated datasets with insertions and deletions. In terms of the computational efficiency, GLaMST runs faster than IgTree.

4.2 Method

A formal description of the BCR lineage tree reconstruction is as follows. Given a set of observed BCR sequences and a root sequence, the maximum parsimony criterion would like to identify the minimum-sized directed tree structure with necessary intermediate sequences, where each directed edge in the tree represents a one-base operation (mutation, insertion, and deletion), and all observed sequences are reachable from the root.

GLaMST reconstructs BCR lineage trees based on the minimum-spanning-tree (MST) [66, 11]. Figure 4.1 shows an outline of this algorithm. We first compute the pairwise edit-distances [67] of the observed sequences including the root node. These pairwise distances reflect the landscape of the observed sequences, in terms of their relative distances and directions with respect to the root node. The algorithm is initialized by considering the root sequence as the root node of the tree, the observed sequences as observed nodes, and no edges. We then grow the tree from the root node by adding directed edges and necessary intermediate nodes toward directions that are more populated by the observed sequences, and iteratively grow the tree until all the observed nodes are reachable from the root node. Observed nodes can either be internal nodes or leaf nodes of the tree, depending on whether they have descendants that are also observed nodes.

4.2.1 Compute edit-distance

The edit-distance from one sequence to another sequence is the size of the minimal set of operations that can convert the first sequence into the second one [67]. In the context of DNA or RNA sequence alignment, one operation is mutation, insertion, or deletion of one

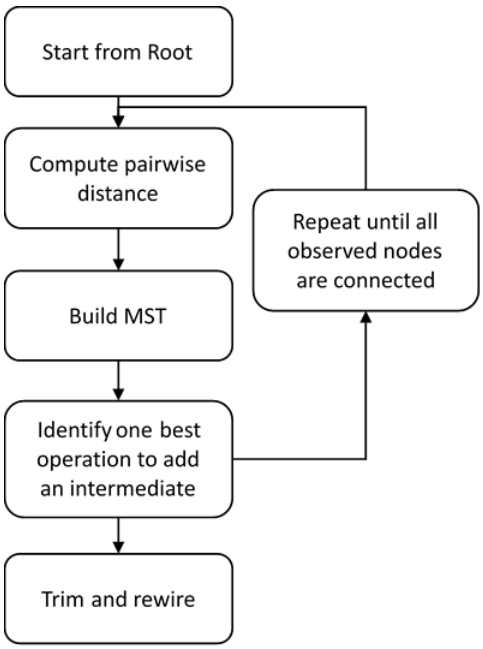

Figure 4.1: **Overview of GLaMST.**

base position. This distance metric is symmetric. We compute the pairwise edit-distances using the Wagner-Fischer algorithm, which is a dynamic programming algorithm with time complexity of $o(mn)$, where m and n are the lengths of the two query sequences [68].

When computing the edit-distance from one sequence to another, we record the one-base operations in the minimal set. If there are multiple minimal sets, we record all operations in those sets, counting each unique operation only once. One example is shown in Figure 4.2. From sequence "ATCCCC" to "GCCCC", the edit distance is 2, because at least 2 one-base operations are needed to convert the first sequence to the second. As shown in Figure 4.2, there exist four paths of length 2 between the two sequences, and therefore, four possible sets of operations corresponding to the edit-distance. Out of the eight operations, four are unique (delete the 1st position, delete the 2nd position, mutate the 1st position to G, mutate the 2nd position to G). These four unique operations are recorded. The recorded operations reflect the "direction" from one sequence to the other, showing what operations can take the first sequence one step toward the second one. This direction information is useful in the next step to choose edges (operations) and intermediate nodes to grow the tree.

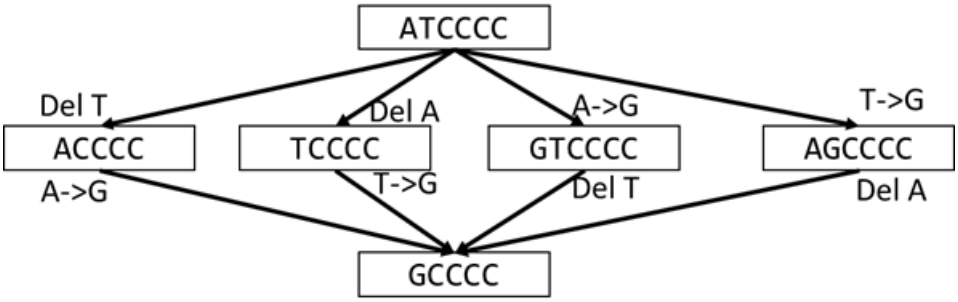

Figure 4.2: **Example of edit-distance.** The edit-distance between these two sequences is 2. There are four sets of operations corresponding to the edit-distance.

Initialize the lineage tree

We initialize GLaMST by treating the root sequence as the root node of the tree, and the observed sequence as other tree nodes. This initial structure does not contain any edge. The root node is considered as the reconstructed part of the lineage tree, whereas all other nodes are standing by, waiting to be brought into the reconstructed part of the lineage tree. Figure 4.3a shows an illustrative example of this initial structure and the distinction between the reconstructed part and the standby nodes.

4.2.2 Iteratively grow the lineage tree

In the first iteration, GLaMST starts by building an MST using the pairwise edit-distances between all nodes. The MST is an undirected tree that connects all nodes with minimum total distances along its edges [11]. An illustrative example is shown in Figure 4.3b. Since the MST is undirected and the edit-distance associated to each edge is often larger than 1, edges of the MST are different from edges of the lineage tree we want to reconstruct. Figure 4.3b uses the dotted undirected lines to represent the MST.

The MST approximates the landscape of the observed sequences with respect to the reconstructed part of the lineage tree, grouping the standby nodes into clusters. For example, in Figure 4.3b, the observed sequences (standby nodes) are divided into three clusters, which locate in three ”directions” from the root node. One cluster consists of nodes {2,

3, 7, 8} in the same direction from the root node, because they are relatively close to each other, and away from the root node and other nodes. There may exist one or several one-base operations that can take the root sequence one step closer to all those four nodes. Such operations are likely to generate intermediate nodes in the maximal parsimony lineage tree.

I then examine clusters of standby nodes originating from the same node in the reconstructed part of the lineage tree, which is all three clusters in Figure 4.3b attached to the root node. We take the sets of operations recorded when computing the edit-distances from the root node to the members in these clusters and count the number of times each operation appears. If one operation appears four times, applying this operation to the root sequence will generate an intermediate sequence that is one step closer to four standby sequences. We choose the operation that appears the most number of times and apply it to the root sequence to generate the first intermediate node to be added to the reconstructed part of the lineage tree. As shown in the illustrative example in Figure 4.3c, the reconstructed part now contains two nodes and one directed edge. The added node is typically along one branch of the MST, representing a common ancestor of the observed nodes in one cluster, but it is also possible that the added node is a common ancestor of multiple clusters.

The second iteration starts with the reconstructed part of the lineage tree and the standby nodes, as shown in Figure 4.3d. The previous MST is discarded, because the newly added node may cause the structure of the MST to change. In this iteration, we rebuild a new MST to connect the standby nodes to the reconstructed part of the lineage tree, as shown in Figure 4.3e. The new MST divides the standby nodes into four clusters. Two clusters originate from the root node, and two clusters originate from node 9. We examine the one-base operations for clusters attached to the same node in the reconstructed part of the tree (operations that take root node R to nodes {4} and {1, 5, 6}, and operations that take node 9 to nodes {2, 8} and {3, 7}), choose the origination-operation pair that appears the most number of times, and apply the operation to the origination node to generate an intermediate node to be added to the reconstructed part of the lineage tree. In the illustrative example in

Figure 4.3f, the chosen origination-operation pair generated an intermediate node from the root node, which is one step closer toward the cluster $\{1, 5, 6\}$.

The subsequent iterations operate exactly the same as the second iteration. When selecting the most frequently appearing origination-operation pair, if there is a tie, we randomly choose one. When the new node suggested by the chosen origination-operation pair is identical to an observed node, the observed node is recruited into the reconstructed part of the lineage tree using a directed edge from the origination node to the observed node, and no intermediate node is added. The iteration continues until all observed nodes are recruited into the reconstructed part of the lineage tree.

4.2.3 Trim and rewire the lineage tree

The lineage tree reconstructed by this heuristic iterative procedure can be reduced by trimming off unnecessary branches. The iterative procedure may occasionally produce branches whose leaf nodes are not observed nodes, because the process of growing the tree is guided by the MST which only approximates the structure of the underlying lineage tree. Branches with unobserved nodes as leaves are unnecessary to explain the mutation process that gives rise to the observed nodes. To trim the lineage tree, we remove all unobserved intermediate and leaf nodes that do not have any observed nodes as descendants.

Another possible improvement is rewiring. In the reconstructed lineage tree, we can detach a subtree by removing the edge pointing to the root of the subtree, reattached it to some other node, and then trim the resulting tree. The trimming operation removes intermediate unobserved nodes right upstream of the removed edge. The reattaching operation introduces additional intermediate nodes if the edit-distance is larger than one between the subtree root and the node it reattaches to. It is possible that such a rewiring operation can reduce the size of the lineage tree. We consider all the observed nodes and branching nodes (i.e. out-degree larger than one) for rewiring. For each node under rewiring consideration, we try to rewire it to all possible nodes in the lineage tree and examine whether we can

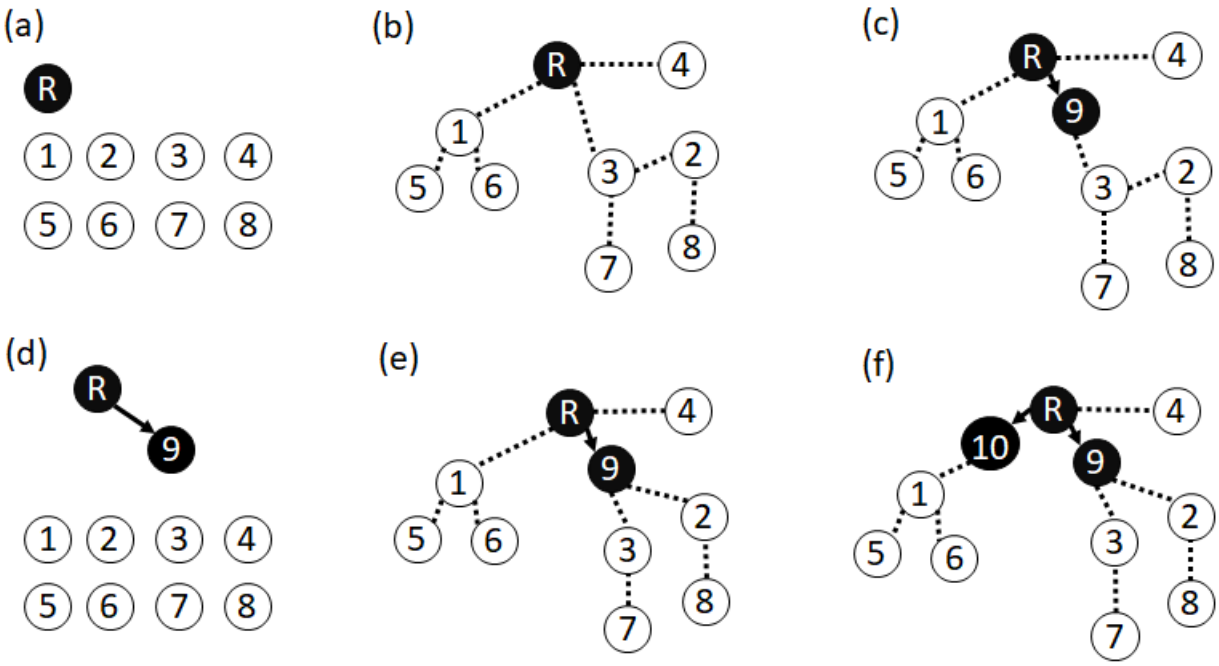

Figure 4.3: **GLaMST tree construction process.** This example shows the first two iterations of an illustrative example. In each visualization, black nodes and solid arrows represent the reconstructed part of the lineage tree. White nodes represent observed sequences that are standing by and waiting to be incorporated into the reconstructed part of the lineage tree. The dotted lines represent the MST, which guides the algorithm in growing the reconstructed part of the lineage tree. (a) GLaMST is initialized by treating the root node as the reconstructed part and all other observed nodes as standing by. (b) An MST is constructed based on the pairwise edit-distance. (c) GLaMST selects the most frequent origination-operation pair to create and insert an intermediate node and grow the reconstructed part. (d-f) The second iteration of the process to insert another node to the reconstructed part.

reduce the tree size. If yes, we accept the rewiring operation and examine the resulting lineage tree again for possible rewiring operations that may further reduce the tree size. We repeat this process until no rewiring operation can reduce the tree size.

4.3 Result

4.3.1 Reconstruct lineage trees using simulated data

To compare IgTree, dnapars and GLaMST, we generated simulated datasets using nine different simulation settings, varying the root sequence length and the relative probabilities

Table 4.1: **Comparison of tree size based on simulated data.**

Simulation	Seq Length	Mutation Distribution (mutation, insertion, deletion)	IgTree				dnapars				GLaMST			
			Larger	Same	Smaller	Success	Larger	Same	Smaller	Success	Larger	Same	Smaller	Success
1	300	(1.00, 0.00, 0.00)	10	469	21	98%	0	480	20	100%	6	473	21	99%
2	300	(0.98, 0.01, 0.01)	12	468	20	98%	3	477	20	99%	7	472	21	99%
3	300	(0.90, 0.05, 0.05)	54	424	22	89%	34	444	22	93%	11	464	25	98%
4	80	(1.00, 0.00, 0.00)	39	390	71	92%	1	420	79	100%	38	391	71	92%
5	80	(0.98, 0.01, 0.01)	71	373	56	86%	20	416	64	96%	28	414	58	94%
6	80	(0.90, 0.05, 0.05)	130	325	45	74%	103	345	52	79%	26	412	62	95%
7	20	(1.00, 0.00, 0.00)	77	234	189	85%	8	241	251	98%	46	251	203	91%
8	20	(0.98, 0.01, 0.01)	116	201	183	77%	54	221	225	89%	35	236	229	93%
9	20	(0.90, 0.05, 0.05)	278	129	93	44%	234	153	113	53%	47	255	198	91%

of mutation, insertion, and deletion as shown in the first column of Table 4.1. Following parameters in a previous simulation study [34], we generated 500 random trees in each simulation setting. The overall tree size ranged from 20 to 80, and the number of observed nodes ranged from 2 to 24. Therefore, each simulated lineage tree contained 20$\sim$80 BCR sequences randomly generated by mutations, insertions or deletions of the root sequence, and the number of observed sequences ranged from 2 to 24. Table 4.1 shows the comparison of the size of lineage trees reconstructed by three different algorithms in all nine simulated datasets.

As shown in Table 4.1, in the first simulation setting where the sequence length was 300 and probabilities of insertion and deletion were 0, all three methods performed well. We recorded how many times the size of the reconstructed tree is smaller, equal, or larger than the simulated ground truth. A reconstructed tree larger than the simulated ground truth was undesirable because the reconstruction failed to achieve the goal of maximum parsimony. A reconstructed tree could occasionally be smaller than the simulated ground truth. This was because the observed sequences represented only a subset of the simulated mutation process, and it was possible to explain such partial observations by trees smaller than the simulated mutation process. We considered it a success if the reconstructed tree was of smaller or equal size compared to the simulated ground truth. In the first simulation setting, the success rates of all three algorithms were $>$98%. This was mainly because the first simulation setting was relatively simple: given a sequence length of 300 and the underlying tree size of 20$\sim$80, the simulated mutations seldom occurred more than once at

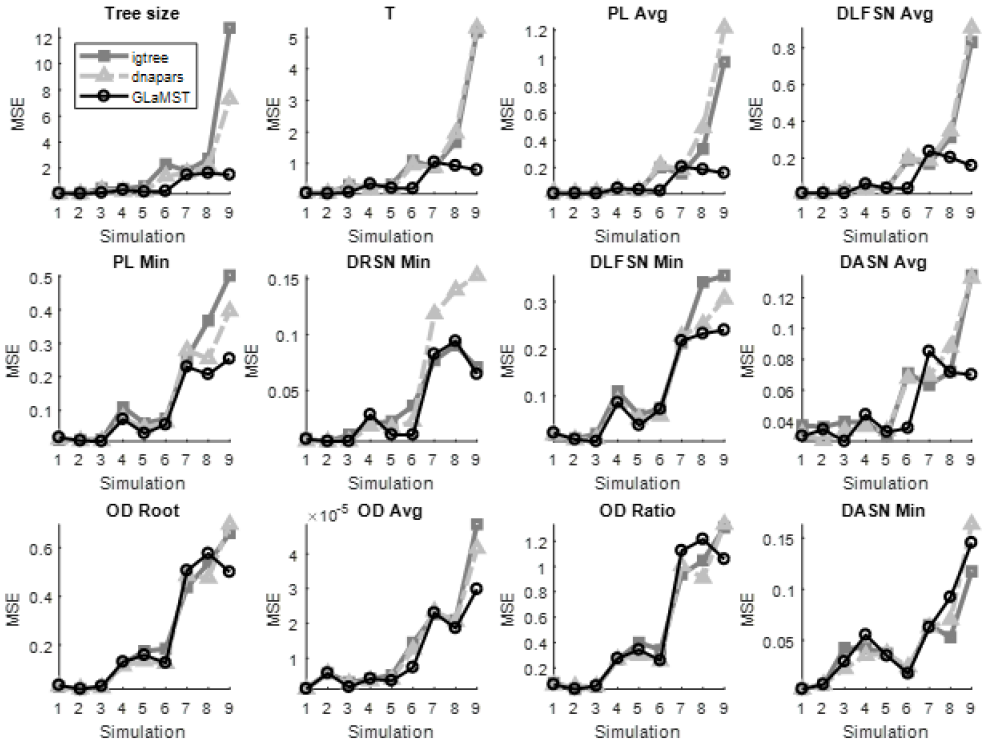

Figure 4.4: **Comparison based on tree features in simulated data.** These figures compare GLaMST, IgTree, and dnapars using 12 tree features. Descriptions of these features are listed in Table 4.2. Each panel corresponds to one tree feature, the x-axis represents the nine simulation settings and the y-axis is the mean square error between reconstructed trees and simulated ground truth. Tree features in the first row suggest that GLaMST is significantly better than the other two methods. The second row shows tree features where GLaMST moderately outperforms the other methods. The third row shows tree features where the three algorithms show similar performance.

the same position.

Comparing the first three simulation settings with the same root sequence length but different proportions of mutations, insertions and deletions, we can see that the performance of all three algorithms decreased with increased insertions and deletions. The same trend was seen in simulation settings with shorter sequence length, showing that the presence of insertions and deletions made the maximum parsimony lineage tree reconstruction problem more challenging.

In the first set of three simulations, the mutations, insertions, and deletions are uniformly distributed in simulated sequences of length 300. In reality, the frequency of mu-

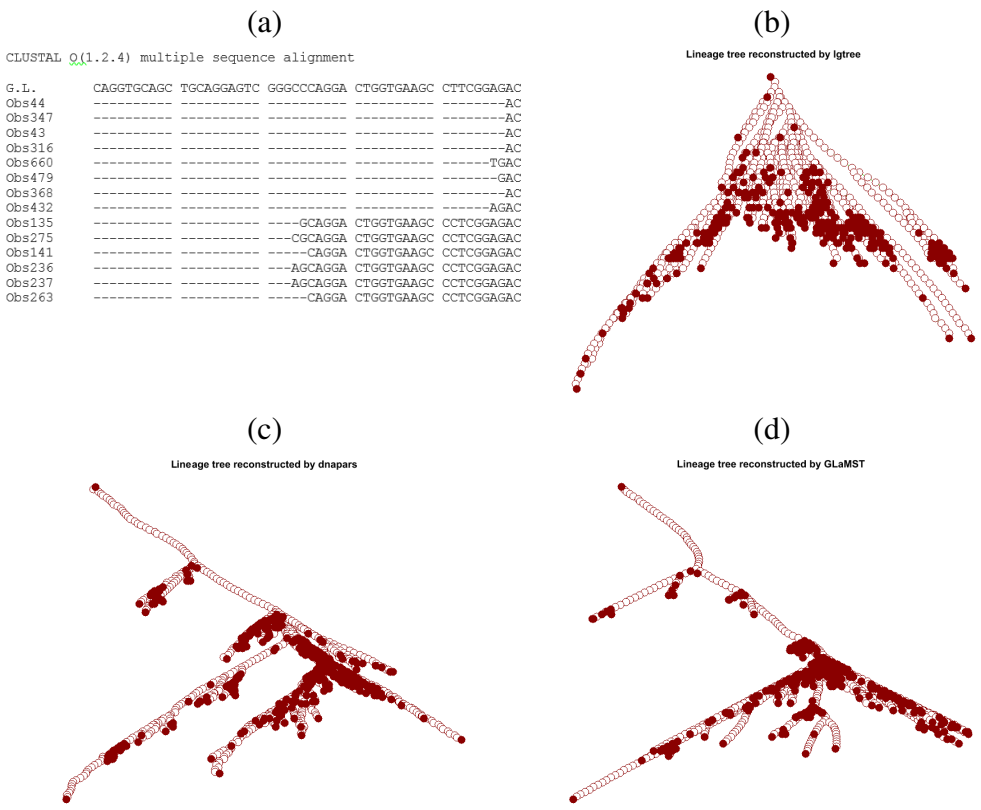

Figure 4.5: **Comparison based on real data.** (a) Multiple sequence alignment of the root node and 14 selected observed nodes. This sequence alignment shows a fixed gap between root node and observed nodes. (b) Lineage tree constructed by Igtree. Square represents the root node. The solid nodes are the observed sequences, while the white/empty nodes are intermediate nodes. (c) Lineage tree constructed by dnapars. (d) Lineage tree constructed by GLaMST.

tations is not uniform across the BCR. For example, the V(D)J region of the BCR can be partitioned into framework regions (FWRs) which typically have lower observed mutation frequencies, and complementarity determining regions (CDRs) which have higher observed mutation frequencies [35]. To examine a simpler situation that has the flavor of variable mutation rate, we created two subsequent sets of simulations with short sequence lengths of 80 and 20, respectively. These simulations were equivalent to simulating the length 300 sequences while constraining the mutations to only a sub-region of length either 80 or 20. These simulation settings with shorter sequence length were progressively more difficult because the simulated mutations, insertions, and deletions in shorter sequences were more likely to occur multiple times at the same position, which represented higher

mutation rates. Table 4.1 shows that the reconstruction performance of all three algorithms consistently decreased with higher mutation rates.

Overall, in terms of tree size shown in Table 4.1, GLaMST and dnapars consistently outperformed IgTree in all nine simulation settings. In simulation settings without insertions and deletions, dnapars outperformed GLaMST by a relatively small margin. However, in simulation settings with insertions and deletions, GLaMST achieved the significantly better performance than dnapars, especially in the last the most challenging simulation setting.

In addition to tree size, we also compared the three algorithms based on tree features defined in the MTree program for lineage tree measurement [69, 70], especially features that are highly correlated with selection pressures [70]. We considered 12 features listed in Table 4.2. For each simulated lineage tree, we computed the difference between the tree features of the simulation ground truth and the tree features of the lineage trees reconstructed by all three algorithms, and then normalized the differences by dividing by the range of corresponding tree features in the simulation ground truth. The average differences for the simulation settings and algorithms are compared in Figure 4.4. For these 12 tree features, GLaMST achieved differences either smaller than or comparable to the other two algorithms, meaning that the lineage trees constructed by GLaMST were more similar to the simulation ground truth compared to the other two algorithms, especially in the most challenging simulation setting.

4.3.2 Reconstruct lineage trees using on BCR sequence data

In order to produce a biological dataset for testing, peripheral blood mononuclear cells (PBMC) were collected from an individual afflicted with Pemphigus Vulgaris. These cells were sorted via fluorescence-activated cell sorting (FACS) into 4 major B cell populations: Naive (CD19+IgD+CD27-), Switched memory (CD19+IgD-CD27+), Double negative (CD19+IgD-CD27-), and Plasmablasts (CD19+/-CD27++CD38++). RNA was extracted and immunoglobulin heavy chain transcripts were amplified using RT-PCR with

Table 4.2: **Comparison of 12 tree features based on real data.**

	Description	IgTree	dnapars	GLaMST
Tree Size	Total number of tree nodes	1396	1199	1190
OD-Root	Out-degree of the root of the tree	1	1	1
OD-Avg	Average out-degree of all tree nodes	0.999	0.999	0.999
OD-Ratio	Ratio between OD-Root and OD-Avg	1.001	1.001	1.001
T	Maximum depth of the tree leaves	62	147	130
PL-Min	Minimum distance from root to any leaf	12	38	36
PL-Avg	Average distance from root to all nodes	30.164	84.624	79.821
DRSN-Min	Minimum distance from root to any branching node	2	36	34
DLFSN-Min	Minimum distance from first branching node to leaves	10	2	2
DLFSN-Avg	Average distance from first branching node to leaves	28.164	48.624	45.821
DASN-Min	Minimum distance between branching nodes	1	1	1
DASN-Avg	Average distance between branching nodes	2.903	2.590	2.216

primers located in the framework region 1 for VH families 1-7 and constant regions for IgM, IgG, and IgA. The amplicons were subsequently sequenced on an Illumina MiSeq using 300bp x2 paired-end reads. Reads were processed using our IgSeq pipeline as described in ([71]), where sequences from all populations were quality filtered, joined, and divided into clones based on same V gene usage, same J gene usage, identical CDR3 length, and 85% CDR3 similarity. Therefore, the data for each individual clone consisted of sequences sharing the same V gene, the same J gene, and the same CDR3 length with 85% sequence similarity in the CDR3 region. Data for individual clones were then used as separate datasets for further testing of GLaMST, with the largest clone being illustrated here.

In the dataset corresponding to the largest clone, the lengths of the observed sequences were around 320, and the total number of observed sequences was 684. The root sequence had a 48-base segment at the five-prime end that did not exist in most of the observed sequences, as shown in Figure 4.5a, which visualized a few selected observed sequences using multiple alignments ([41]). Therefore, a fixed size gap existed between root and all the observed sequences. Although this gap was due to the location of the primer, it should be recognized by the lineage tree reconstruction algorithms.

The lineage tree reconstructed by IgTree was of size 1396 (Figure 4.5b). dnapars generated a lineage tree which has 1199 nodes (Figure 4.5c). The lineage tree reconstructed by GLaMST contained 1190 nodes (Figure 4.5d). Therefore, GLaMST and dnapars significantly outperformed IgTree in achieving the maximum parsimony criterion. From Figures 4.5c and 4.5d, we can see that the trees reconstructed by GLaMST and dnapars shared similar topology, with a long chain of intermediate nodes off of the root before the first branching point. This chain corresponded to the 48-base segment in the root sequence which did not exist in the observed sequences. In contrast, Figure 4.5b shows that IgTree failed to recognize this common difference between the observed sequences and the root, generating a tree wider and shallower tree with much larger size compared to GLaMST and dnapars. We also computed the 12 tree features in Table 4.2, which confirmed that the topology of the GlaMST and dnapars results were similar to each other but very different from that of IgTree.

This dataset contained 684 observed sequences, which was much larger than the simulated data, and therefore, enabled us to compare the running time of the three algorithms. These algorithms were compared on a desktop computer with Intel i7-3770 processor at 3.40GHz and 32GM memory. dnapars spent roughly 5 days to reconstruct a lineage tree for this dataset. IgTree took 20 minutes, while GLaMST took only 16 minutes. Although dnapars and GLaMST reconstructed similar tree structures, GLaMST is significantly more efficient computationally.

4.4 Discussion

High throughput sequencing of BCR repertoire analysis motivated the computational question of reconstructing lineage trees based on partially observed tree nodes. We developed the GLaMST algorithm to reconstruct lineage trees with maximum parsimony. As suggested by its name, GLaMST grows lineage trees along the minimum spanning tree, which is a simple and efficient heuristic algorithm. Using both simulated and real data, we demon-

strated that GLaMST is more effective in achieving maximum parsimony compared two existing algorithms. GLaMST is also computationally more efficient, enabling its application in analysis of large BCR sequencing datasets. Integrating GLaMST into existing BCR sequencing analysis frameworks can lead to significant improves in the lineage tree reconstruction aspect of the analysis.

Conclusion and Outlook

My research focuses on developing new methods to solve difficulties in single cell analysis. In dealing with the interpretation problems in the data-driven analysis of flow/mass cytometry data, I first developed an algorithm named Cluster-to-Gate(C2G) to generate a gating hierarchy of nested two-dimensional gates from populations defined by either manual gating analysis, automated clustering algorithms, or visualization-based methods. C2G fills the gap between high-dimensional analysis algorithms for cytometry data and the conventional gating visualization that biologists are accustomed to, which will help to promote the adoption of high-dimensional analysis algorithms. In addition, the hierarchical representation can also benefit algorithm developers, assisting them to visualize their clustering results for debugging and parameter tuning. Another potential application of C2G is to derive a gating hierarchy for completely unlabeled data. Upon using any general clustering methods (for example k-means or its variants), we can apply C2G to obtain a gating hierarchy to explain the relationship among the clusters. Such a hierarchical explanation is typically not available in high-dimensional clustering algorithms, except for RchyOptimyx [24]. An advantage of this application is that it is flexible and easy to interpret because the gating hierarchy reveals which marker pairs are important in separating which subsets of populations. C2G can also be used to perform panel refinement. The average F-score from C2G by ignoring one of the protein markers can serve as a measurement of the relative importance of the protein marker. By gradually removing the least informative markers, C2G may reveal that only a subset of the protein markers in the staining panel can effectively explain cell clusters defined by all the protein markers in the entire panel. The limitation of C2G is that the method does not properly address other difficulties of data-driven methods including identifying the hierarchical structure and handling unbalanced population size.

To overcome the weakness of C2G, I initiated the knowledge-graph approach to analyze flow/mass cytometry data. For this project, the most troubling bottleneck is to collect

enough high-quality data. The data is collected by Google image search which might not cover all important gating strategies as Google images search only present images that relevant to the keywords, not the research topics. In addition, we only use the cell ontology database to match cell types. This database is not comprehensive, and many convention names cannot be correctly matched. To address such two weaknesses, more human labor should be adapted in collecting data and matching cell types. As many flow/mass cytometry data are publicly available and the gating strategies are detailed in the corresponding publication, researchers can read the papers one by one and manually curate enough number of publications.

In dealing with the lineage tree reconstruction problem, I developed a new algorithm named GLaMST that outperforms state-of-art in both accuracy and efficiency. The main improvement of GLaMST is that it adapts minimum spanning tree, a more efficient heuristic, to approximate the best solution in achieving maximum parsimony. However, the maximum parsimony criterion itself is an approximation of the real tree. It assumes that different mutations happen uniformly, which is usually not true. Researches show that the mutations distribution highly depended on the context [72] and selection pressure [73]. To incorporate such information, one future improvement of GLaMST is to replace the maximum parsimony criterion with maximum likelihood or Bayesian estimation. In the step of deciding which branch to insert new mutation, the current implementation of GLaMST picks up the best branch only by how much overall tree size can be reduced. In the future version, this measurement can be weighted by how likely each mutation can happen according to the constructed part of the tree. To achieve such a goal, a decent amount of prior knowledge is required. Such prior knowledge can be collected by summarizing information from various publications.

All of the potential improvement of the above projects involve a decent level of data collection. Such a data-collection process is also labor-intensive and time-consuming. To improve working efficiency, a semi-automatically way to help to collect publications in

the target area and extracting common messages from massive publications would have a bright future. Of course, building a framework of artificial intelligence that can actually understand scientific publications like the human brain is impossible today. However, massive researches on social network study [74] and recent breakthrough in natural language processing with deep learning [75] make it possible to efficiently collect and organize publications according to the area of interests. Citation network is a well-studied social network that can help researchers identify subfields of research and recognize main publications in certain subfields [76]. Techniques in such research can be used to help researchers quickly find massive publications in the area of interest, for example, flow cytometry or B cell somatic hypermutation. Upon obtaining the list of publications, natural-language-processing techniques can help researchers to classify the publications by features including organism/tissue of research, experimental protocol, and whether raw data are provided. Turning these techniques into a mature tool can also help junior students to quickly start their research. Anticipated difficulties in achieving this goal are listed below. First, automatically access the massive number of publications might lead to copyright issue. Second, parsing publications from different websites/sources might suffer from formats discrepancy. Third, since a completely random search can also give a list of publications, a proper way to evaluate the obtained publications is vitally important to make it useful.

REFERENCES

[1] J. E. Darnell, H. F. Lodish, D. Baltimore, *et al.*, *Molecular cell biology*. Scientific American Books New York, 1990, vol. 2.

[2] D. Wang and S. Bodovitz, "Single cell analysis: The new frontier in omics," *Trends in biotechnology*, vol. 28, no. 6, pp. 281–290, 2010.

[3] N. M. Toriello, E. S. Douglas, N. Thaitrong, S. C. Hsiao, M. B. Francis, C. R. Bertozzi, and R. A. Mathies, "Integrated microfluidic bioprocessor for single-cell gene expression analysis," *Proceedings of the National Academy of Sciences*, vol. 105, no. 51, pp. 20 173–20 178, 2008.

[4] X. Liu, F. Long, H. Peng, S. J. Aerni, M. Jiang, A. Sánchez-Blanco, J. I. Murray, E. Preston, B. Mericle, S. Batzoglou, *et al.*, "Analysis of cell fate from single-cell gene expression profiles in c. elegans," *Cell*, vol. 139, no. 3, pp. 623–633, 2009.

[5] M. B. Elowitz, A. J. Levine, E. D. Siggia, and P. S. Swain, "Stochastic gene expression in a single cell," *Science*, vol. 297, no. 5584, pp. 1183–1186, 2002.

[6] A. Eldar and M. B. Elowitz, "Functional roles for noise in genetic circuits," *Nature*, vol. 467, no. 7312, p. 167, 2010.

[7] H. Maamar, A. Raj, and D. Dubnau, "Noise in gene expression determines cell fate in bacillus subtilis," *Science*, vol. 317, no. 5837, pp. 526–529, 2007.

[8] N. Baumgarth and M. Roederer, "A practical approach to multicolor flow cytometry for immunophenotyping," *Journal of immunological methods*, vol. 243, no. 1, pp. 77–97, 2000.

[9] S. C. Bendall, E. F. Simonds, P. Qiu, D. A. El-ad, P. O. Krutzik, R. Finck, R. V. Bruggner, R. Melamed, A. Trejo, and O. I. Ornatsky, "Single-cell mass cytometry of differential immune and drug responses across a human hematopoietic continuum," *Science*, vol. 332, no. 6030, pp. 687–696, 2011.

[10] P. K. Chattopadhyay, C.-M. Hogerkorp, and M. Roederer, "A chromatic explosion: The development and future of multiparameter flow cytometry," *Immunology*, vol. 125, no. 4, pp. 441–449, 2008.

[11] P. Qiu, E. F. Simonds, S. C. Bendall, K. D. Gibbs Jr, R. V. Bruggner, M. D. Linderman, K. Sachs, G. P. Nolan, and S. K. Plevritis, "Extracting a cellular hierarchy from

high-dimensional cytometry data with spade," *Nature biotechnology*, vol. 29, no. 10, pp. 886–891, 2011.

[12] Y. Lu, Q. Xue, M. R. Eisele, E. S. Sulistijo, K. Brower, L. Han, E.-a. D. Amir, D. Peer, K. Miller-Jensen, and R. Fan, "Highly multiplexed profiling of single-cell effector functions reveals deep functional heterogeneity in response to pathogenic ligands," *Proceedings of the National Academy of Sciences*, vol. 112, no. 7, E607–E615, 2015.

[13] K. E. Diggins, P. B. Ferrell Jr, and J. M. Irish, "Methods for discovery and characterization of cell subsets in high dimensional mass cytometry data," *Methods*, vol. 82, pp. 55–63, 2015.

[14] L. Han, P. Qiu, Z. Zeng, J. L. Jorgensen, D. H. Mak, J. K. Burks, W. Schober, T. J. McQueen, J. Cortes, S. D. Tanner, *et al.*, "Single-cell mass cytometry reveals intracellular survival/proliferative signaling in flt3-itd-mutated aml stem/progenitor cells," *Cytometry Part A*, vol. 87, no. 4, pp. 346–356, 2015.

[15] P. Qiu, "Computational prediction of manually gated rare cells in flow cytometry data," *Cytometry Part A*, vol. 87, no. 7, pp. 594–602, 2015.

[16] J. M. Irish, J. H. Myklebust, A. A. Alizadeh, R. Houot, J. P. Sharman, D. K. Czerwinski, G. P. Nolan, and R. Levy, "B-cell signaling networks reveal a negative prognostic human lymphoma cell subset that emerges during tumor progression," *Proceedings of the National Academy of Sciences*, vol. 107, no. 29, pp. 12 747–12 754, 2010.

[17] P. Qiu, "Inferring phenotypic properties from single-cell characteristics," *PLoS One*, vol. 7, no. 5, e37038, 2012.

[18] E. Lugli, M. Roederer, and A. Cossarizza, "Data analysis in flow cytometry: The future just started," *Cytometry Part A*, vol. 77, no. 7, pp. 705–713, 2010.

[19] N. Aghaeepour, G. Finak, H. Hoos, T. R. Mosmann, R. Brinkman, R. Gottardo, R. H. Scheuermann, F. Consortium, and D. Consortium, "Critical assessment of automated flow cytometry data analysis techniques," *Nature methods*, vol. 10, no. 3, pp. 228–238, 2013.

[20] Y. Saeys, S. Van Gassen, and B. N. Lambrecht, "Computational flow cytometry: Helping to make sense of high-dimensional immunology data," *Nature Reviews Immunology*, vol. 16, no. 7, pp. 449–462, 2016.

[21] P. Kvistborg, C. Gouttefangeas, N. Aghaeepour, A. Cazaly, P. K. Chattopadhyay, C. Chan, J. Eckl, G. Finak, S. R. Hadrup, and H. T. Maecker, "Thinking outside

the gate: Single-cell assessments in multiple dimensions," *Immunity*, vol. 42, no. 4, pp. 591–592, 2015.

[22] E.-a. D. Amir, K. L. Davis, M. D. Tadmor, E. F. Simonds, J. H. Levine, S. C. Bendall, D. K. Shenfeld, S. Krishnaswamy, G. P. Nolan, and D. Pe'er, "Visne enables visualization of high dimensional single-cell data and reveals phenotypic heterogeneity of leukemia," *Nature biotechnology*, vol. 31, no. 6, pp. 545–552, 2013.

[23] N. Aghaeepour, P. K. Chattopadhyay, A. Ganesan, K. O'neill, H. Zare, A. Jalali, H. H. Hoos, M. Roederer, and R. R. Brinkman, "Early immunologic correlates of hiv protection can be identified from computational analysis of complex multivariate t-cell flow cytometry assays," *Bioinformatics*, vol. 28, no. 7, pp. 1009–1016, 2012.

[24] N. Aghaeepour, A. Jalali, K. O'Neill, P. K. Chattopadhyay, M. Roederer, H. H. Hoos, and R. R. Brinkman, "Rchyoptimyx: Cellular hierarchy optimization for flow cytometry," *Cytometry Part A*, vol. 81, no. 12, pp. 1022–1030, 2012.

[25] S. Meehan, G. Walther, W. Moore, D. Orlova, C. Meehan, D. Parks, E. Ghosn, M. Philips, E. Mitsunaga, and J. Waters, "Autogate: Automating analysis of flow cytometry data," *Immunologic research*, vol. 58, no. 2-3, pp. 218–223, 2014.

[26] G. Finak, J. Frelinger, W. Jiang, E. W. Newell, J. Ramey, M. M. Davis, S. A. Kalams, S. C. De Rosa, and R. Gottardo, "Opencyto: An open source infrastructure for scalable, robust, reproducible, and automated, end-to-end flow cytometry data analysis," *PLoS Comput Biol*, vol. 10, no. 8, e1003806, 2014.

[27] F. Tang, C. Barbacioru, Y. Wang, E. Nordman, C. Lee, N. Xu, X. Wang, J. Bodeau, B. B. Tuch, A. Siddiqui, *et al.*, "Mrna-seq whole-transcriptome analysis of a single cell," *Nature methods*, vol. 6, no. 5, p. 377, 2009.

[28] M. Morey, A. Fernández-Marmiesse, D. Castiñeiras, J. M. Fraga, M. L. Couce, and J. A. Cocho, "A glimpse into past, present, and future dna sequencing," *Molecular genetics and metabolism*, vol. 110, no. 1, pp. 3–24, 2013.

[29] B. Hwang, J. H. Lee, and D. Bang, "Single-cell rna sequencing technologies and bioinformatics pipelines," *Experimental & molecular medicine*, vol. 50, no. 8, p. 96, 2018.

[30] J. J. Calis and B. R. Rosenberg, "Characterizing immune repertoires by high throughput sequencing: strategies and applications," *Trends Immunol.*, vol. 35, no. 12, pp. 581–590, 2014.

[31] L. He, D. Sok, P. Azadnia, J. Hsueh, E. Landais, M. Simek, W. C. Koff, P. Poignard, D. R. Burton, and J. Zhu, "Toward a more accurate view of human B-cell repertoire

by next-generation sequencing, unbiased repertoire capture and single-molecule barcoding," *Sci Rep*, vol. 4, p. 6778, 2014.

[32] G. Georgiou, G. C. Ippolito, J. Beausang, C. E. Busse, H. Wardemann, and S. R. Quake, "The promise and challenge of high-throughput sequencing of the antibody repertoire," *Nat. Biotechnol.*, vol. 32, no. 2, pp. 158–168, 2014.

[33] D. K. Dunn-Walters, A. Belelovsky, H. Edelman, M. Banerjee, and R. Mehr, "The dynamics of germinal centre selection as measured by graph-theoretical analysis of mutational lineage trees," *Dev. Immunol.*, vol. 9, no. 4, pp. 233–243, 2002.

[34] M. Barak, N. S. Zuckerman, H. Edelman, R. Unger, and R. Mehr, "IgTree: creating Immunoglobulin variable region gene lineage trees," *J. Immunol. Methods*, vol. 338, no. 1-2, pp. 67–74, 2008.

[35] G. Yaari and S. H. Kleinstein, "Practical guidelines for B-cell receptor repertoire sequencing analysis," *Genome Med*, vol. 7, p. 121, 2015.

[36] Z. Yang and B. Rannala, "Molecular phylogenetics: principles and practice," *Nat. Rev. Genet.*, vol. 13, no. 5, pp. 303–314, 2012.

[37] A. V. Werhli and D. Husmeier, "Gene regulatory network reconstruction by Bayesian integration of prior knowledge and/or different experimental conditions," *J Bioinform Comput Biol*, vol. 6, no. 3, pp. 543–572, 2008.

[38] S. Whelan, P. Lio, and N. Goldman, "Molecular phylogenetics: state-of-the-art methods for looking into the past," *Trends Genet.*, vol. 17, no. 5, pp. 262–272, 2001.

[39] W. H. Day, D. S. Johnson, and D. Sankoff, "The computational complexity of inferring rooted phylogenies by parsimony," *Mathematical Biosciences*, vol. 81, no. 1, pp. 33–42, 1986.

[40] B. Chor and T. Tuller, "Maximum likelihood of evolutionary trees is hard," in *Research in Computational Molecular Biology: 9th Annual International Conference, RECOMB 2005, Cambridge, MA, USA, May 14-18, 2005. Proceedings*, S. Miyano, J. Mesirov, S. Kasif, S. Istrail, P. A. Pevzner, and M. Waterman, Eds. Springer Berlin Heidelberg, 2005, pp. 296–310.

[41] J. D. Thompson, D. G. Higgins, and T. J. Gibson, "CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice," *Nucleic Acids Res.*, vol. 22, no. 22, pp. 4673–4680, 1994.

[42] D. Sok, U. Laserson, J. Laserson, Y. Liu, F. Vigneault, J. P. Julien, B. Briney, A. Ramos, K. F. Saye, K. Le, A. Mahan, S. Wang, M. Kardar, G. Yaari, L. M. Walker,

B. B. Simen, E. P. St John, P. Y. Chan-Hui, K. Swiderek, S. H. Kleinstein, S. H. Kleinstein, G. Alter, M. S. Seaman, A. K. Chakraborty, D. Koller, I. A. Wilson, G. M. Church, D. R. Burton, and P. Poignard, "The effects of somatic hypermutation on neutralization and binding in the PGT121 family of broadly neutralizing HIV antibodies," *PLoS Pathog.*, vol. 9, no. 11, e1003754, 2013.

[43] M. R. Green, A. J. Gentles, R. V. Nair, J. M. Irish, S. Kihira, C. L. Liu, I. Kela, E. S. Hopmans, J. H. Myklebust, H. Ji, S. K. Plevritis, R. Levy, and A. A. Alizadeh, "Hierarchy in somatic mutations arising during genomic evolution and progression of follicular lymphoma," *Blood*, vol. 121, no. 9, pp. 1604–1611, 2013.

[44] N. T. Gupta, J. A. Vander Heiden, M. Uduman, D. Gadala-Maria, G. Yaari, and S. H. Kleinstein, "Change-O: a toolkit for analyzing large-scale B cell immunoglobulin repertoire sequencing data," *Bioinformatics*, vol. 31, no. 20, pp. 3356–3358, 2015.

[45] J. Felsenstein, "PHYLIP - phylogeny inference package (version 3.2)," *Cladistics*, vol. 5, pp. 164–166, 1989.

[46] J. N. Stern, G. Yaari, J. A. Vander Heiden, G. Church, W. F. Donahue, R. Q. Hintzen, A. J. Huttner, J. D. Laman, R. M. Nagra, A. Nylander, D. Pitt, S. Ramanan, B. A. Siddiqui, F. Vigneault, S. H. Kleinstein, D. A. Hafler, and K. C. O'Connor, "B cells populating the multiple sclerosis brain mature in the draining cervical lymph nodes," *Sci Transl Med*, vol. 6, no. 248, 248ra107, 2014.

[47] S. Krishnaswamy, M. H. Spitzer, M. Mingueneau, S. C. Bendall, O. Litvin, E. Stone, D. Peer, and G. P. Nolan, "Conditional density-based analysis of t cell signaling in single-cell data," *Science*, vol. 346, no. 6213, p. 1 250 689, 2014.

[48] M. H. Spitzer, P. F. Gherardini, G. K. Fragiadakis, N. Bhattacharya, R. T. Yuan, A. N. Hotson, R. Finck, Y. Carmi, E. R. Zunder, and W. J. Fantl, "An interactive reference framework for modeling a dynamic immune system," *Science*, vol. 349, no. 6244, p. 1 259 425, 2015.

[49] N. Aghaeepour, R. Nikolic, H. H. Hoos, and R. R. Brinkman, "Rapid cell population identification in flow cytometry data," *Cytometry Part A*, vol. 79, no. 1, pp. 6–13, 2011.

[50] S. Van Gassen, B. Callebaut, M. J. Van Helden, B. N. Lambrecht, P. Demeester, T. Dhaene, and Y. Saeys, "Flowsom: Using self-organizing maps for visualization and interpretation of cytometry data," *Cytometry Part A*, vol. 87, no. 7, pp. 636–645, 2015.

[51] J. H. Levine, E. F. Simonds, S. C. Bendall, K. L. Davis, D. A. El-ad, M. D. Tadmor, O. Litvin, H. G. Fienberg, A. Jager, and E. R. Zunder, "Data-driven phenotypic

dissection of aml reveals progenitor-like cells that correlate with prognosis," *Cell*, vol. 162, no. 1, pp. 184–197, 2015.

[52] J. Spidlen, W. Moore, D. Parks, M. Goldberg, C. Bray, P. Bierre, P. Gorombey, B. Hyun, M. Hubbard, and S. Lange, "Data file standard for flow cytometry, version fcs 3.1," *Cytometry Part A*, vol. 77, no. 1, pp. 97–100, 2010.

[53] Y. Ge and S. C. Sealfon, "Flowpeaks: A fast unsupervised clustering for flow cytometry data via k-means and density peak finding," *Bioinformatics*, vol. 28, no. 15, pp. 2052–2058, 2012.

[54] L. Danon, A. Diaz-Guilera, J. Duch, and A. Arenas, "Comparing community structure identification," *Journal of Statistical Mechanics: Theory and Experiment*, vol. 2005, no. 09, P09008, 2005.

[55] J. Spidlen, K. Breuer, C. Rosenberg, N. Kotecha, and R. R. Brinkman, "Flowrepository: A resource of annotated flow cytometry datasets associated with peer-reviewed publications," *Cytometry Part A*, vol. 81, no. 9, pp. 727–731, 2012.

[56] T. J. Chen and N. Kotecha, "Cytobank: Providing an analytics platform for community cytometry data analysis and collaboration," in *High-Dimensional Single Cell Analysis*, Springer, 2014, pp. 127–157.

[57] G. Finak, W. Jiang, J. Pardo, A. Asare, and R. Gottardo, "Qualifier: An automated pipeline for quality assessment of gated flow cytometry data," *BMC bioinformatics*, vol. 13, no. 1, p. 252, 2012.

[58] J. Bard, S. Y. Rhee, and M. Ashburner, "An ontology for cell types," *Genome biology*, vol. 6, no. 2, R21, 2005.

[59] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, pp. 580–587.

[60] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 779–788.

[61] M. Malek, M. J. Taghiyar, L. Chong, G. Finak, R. Gottardo, and R. R. Brinkman, "Flowdensity: Reproducing manual gating of flow cytometry data by automated density-based cell population identification," *Bioinformatics*, vol. 31, no. 4, pp. 606–607, 2014.

[62] P. Y. Simard, D. Steinkraus, and J. C. Platt, "Best practices for convolutional neural networks applied to visual document analysis," in *null*, IEEE, 2003, p. 958.

[63] A. L. Maas, A. Y. Hannun, and A. Y. Ng, "Rectifier nonlinearities improve neural network acoustic models," in *Proc. icml*, vol. 30, 2013, p. 3.

[64] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167*, 2015.

[65] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[66] J. B. Kruskal, "On the shortest spanning subtree of a graph and the traveling salesman problem," *Proceedings of the American Mathematical Society*, vol. 7, no. 1, pp. 48–50, 1956.

[67] R. C. Prim, "Shortest connection networks and some generalizations," *Bell System Technology Journal*, vol. 36, pp. 1389–1401, 1957.

[68] R. A. Wagner and M. J. Fischer, "The string-to-string correction problem," *J. ACM*, vol. 21, no. 1, pp. 168–173, 1974.

[69] D. K. Dunn-Walters, H. Edelman, and R. Mehr, "Immune system learning and memory quantified by graphical analysis of B-lymphocyte phylogenetic trees," *BioSystems*, vol. 76, no. 1-3, pp. 141–155, 2004.

[70] M. Uduman, M. J. Shlomchik, F. Vigneault, G. M. Church, and S. H. Kleinstein, "Integrating B cell lineage information into statistical tests for detecting selection in Ig sequences," *J. Immunol.*, vol. 192, no. 3, pp. 867–874, 2014.

[71] C. M. Tipton, C. F. Fucile, J. Darce, A. Chida, T. Ichikawa, I. Gregoretti, S. Schieferl, J. Hom, S. Jenks, R. J. Feldman, R. Mehr, C. Wei, F. E. Lee, W. C. Cheung, A. F. Rosenberg, and I. Sanz, "Diversity, cellular origin and autoreactivity of antibody-secreting cell population expansions in acute systemic lupus erythematosus," *Nat. Immunol.*, vol. 16, no. 7, pp. 755–765, 2015.

[72] B. T. Messmer, E. Albesiano, D. Messmer, and N. Chiorazzi, "The pattern and distribution of immunoglobulin vh gene mutations in chronic lymphocytic leukemia b cells are consistent with the canonical somatic hypermutation process," *Blood*, vol. 103, no. 9, pp. 3490–3495, 2004.

[73] K. B. Hoehn, A. Fowler, G. Lunter, and O. G. Pybus, "The diversity and molecular evolution of b-cell receptors during infection," *Molecular biology and evolution*, vol. 33, no. 5, pp. 1147–1157, 2016.

[74] S. Fortunato, "Community detection in graphs," *Physics reports*, vol. 486, no. 3-5, pp. 75–174, 2010.

[75] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, p. 436, 2015.

[76] P Chen and S. Redner, "Community structure of the physical review citation network," *Journal of Informetrics*, vol. 4, no. 3, pp. 278–290, 2010.